IBM Tivoli Access Manager

Plug-in for Web Servers User's Guide

Version 3.9

GC23-4686-00

IBM Tivoli Access Manager

Plug-in for Web Servers User's Guide

Version 3.9

GC23-4686-00

Note
Before using this information and the —product it supports, read the information in Appendix D, “Notices” on page 109.

First Edition (April 2002)

This edition applies to version 3.9, of IBM Tivoli Access Manager: Plug-in for Web Servers (product number 5724–C08) and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 2002. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures vii

Tables ix

Preface xi
Who should read this book xi
What this book contains xi
Publications xii
IBM Tivoli Access Manager xii
Related publications xv
Accessing publications online xvi
Ordering publications. xvi
Providing feedback about publications xvii
Accessibility xvii
Contacting customer support xvii
Conventions used in this book xvii
Typeface conventions xvii

Chapter 1. Introducing IBM Tivoli Access Manager Plug-in for Web Servers 1
Understanding Access Manager Plug-in for Web Servers technology 1
Basic operational components and architecture 1
Support for virtual hosts 2
Protecting your Web space with Access Manager Plug-in for Web Servers 3
Planning and implementing security policy 3
Understanding Access Manager Plug-in for Web Servers authentication. 4
The goals of authentication 4
Understanding credential acquisition 5
The Extended Privilege Attribute Certificate (EPAC) 5

Chapter 2. Installing IBM Tivoli Access Manager Plug-in for Web Servers 7
Supported platforms. 7
Disk and memory requirements 7
Software prerequisites 7
Installing Access Manager Plug-in for Web Servers 8
Installing the plug-in on AIX – IHS 8
Installing the plug-in on the Solaris Operating Environment – iPlanet 10
Installing the plug-in on Windows – IIS 11
Removing Access Manager Plug-in for Web Servers 12
Removing the plug-in from Windows – IIS 12
Removing the plug-in from AIX – IHS 13
Removing the plug-in from Solaris Operating Environment – iPlanet 13

Chapter 3. IBM Tivoli Access Manager Plug-in for Web Servers configuration 15
General plug-in information 15
Introducing the pdwebpi.conf configuration file 15
The pdwebpimgr.conf configuration file 16
Root directory of the Access Manager Plug-in for Web Servers installation 16
Starting and stopping Access Manager Plug-in for Web Servers 16
HTTP error messages 17
Configuring the Authorization Server. 18
Configuring Worker Threads 18
Setting the Maximum Session Lifetime 18
Configuring Error Pages 19
Configuring for virtual host servers 20

© Copyright IBM Corp. 2002

Web server specific configuration 22
Configuring plug-in auditing, logging, tracing and the cache database. 24
Understanding audit records 24
Auditing configuration 26
Tracing Plug-in actions 26
Cache database settings 27
Configuring the authorization API service 27

Chapter 4. IBM Tivoli Access Manager Plug-in for Web Servers authentication 29
Understanding the authentication process 29
Configuring authentication 32
Managing session state 37
Configuring the plug-in session/credentials cache 38
Maintaining session state with the SSL session ID 40
Maintaining session state using Basic Authentication 40
Maintaining session state with Session Cookies 41
Maintaining session state using HTTP headers. 42
Maintaining session state using IP addresses 42
Managing authentication parameters 43
Authentication configuration overview 43
Configuring Basic Authentication 46
Configuring forms authentication 47
Configuring certificate authentication. 48
Configuring token authentication 50
Configuring failover cookie authentication 51
Configuring IV header authentication 53
Configuring HTTP header authentication 55
Configuring IP address authentication 56
Configuring tag value post-authorization processing. 57
Setting authentication parameters for virtual hosts 58
Supporting Multiplexing Proxy Agents (MPA). 59
Valid session data types and authentication methods 59
Authentication process flow for MPA and multiple clients 60
Enabling MPA authentication 61
Create a user account for the MPA 62
Add the MPA account to the pdwebpi-mpa-servers group 62

Chapter 5. IBM Tivoli Access Manager Plug-in for Web Servers security policy 63
Plug-in specific Access Control List (ACL) policies 63
/PDWebPI/*host* or *virtual_host* 63
Plug-in ACL permissions 63
Default /PDWebPI ACL policy 64
Three strikes logon policy 65
Command syntax 65
Password strength policy 66
Password strength policy set by the pdadmin utility. 66
Specific user and global settings 68
Authentication strength Protected Object Policy (Step-up) 68
Configuring levels for step-up authentication 69
Enabling step-up authentication 70
Step-up authentication notes and limitations 71
Reauthentication Protected Object Policy 71
Conditions affecting POP reauthentication 71
Creating and applying the reauthentication POP 72
Network-based authentication Protected Object Policy 73
Specifying IP addresses and ranges 73
Disabling step-up authentication by IP address 74
Network-based authentication algorithm 74
Quality of protection Protected Object Policy 74
Handling unauthenticated users (HTTP/HTTPS) 75

Processing a request from an anonymous client . . . 75
Forcing user log on . . . 75
Applying unauthenticated HTTPS . . . 76
Controlling unauthenticated users with ACL/POP policies . . . 76

Chapter 6. Web single signon solutions . . . 77
Single signon concepts . . . 77
Automatically signing-on to a secured application . . . 77
Configuring single signon to secure applications using HTTP headers . . . 78
Single signon to WebSphere application server using LTPA cookies . . . 79
Single signon to the plug-in from WebSEAL or other proxy . . . 80
Configuring IV header single signon to Access Manager Plug-in for Web Servers . . . 80
Using the Failover cookie for single signon . . . 81
Enabling single signon using Failover cookies . . . 81
Configuring Failover cookie parameters . . . 81

Chapter 7. e-Community single signon . . . 83
Introducing e-community single signon . . . 83
e-Community single signon features and requirements . . . 84
e-Community single signon process flow . . . 84
Understanding the e-community cookie . . . 86
Understanding the ″Vouch For″ request and reply . . . 86
The ″vouch for″ request . . . 86
The ″vouch for″ reply . . . 86
Understanding the ″Vouch For″ token . . . 87
Encrypting the ″Vouch For″ token . . . 87
Configuring an e-community . . . 88
Configuring e-community single signon - an example . . . 90

Appendix A. pdwebpi.conf reference . . . 93

Appendix B. Authentication methods quick reference . . . 103

Appendix C. Command quick reference . . . 107

Appendix D. Notices . . . 109
Trademarks . . . 110

Index . . . 113

Figures

1. Plug-in and Access Manager component interaction. 2
2. Web server access decisions. 31
3. Plug-in process flow for determining authentication module. 35
4. Authentication challenge process logic. 36
5. Plug-in process flow for determining session module. 38
6. Typical server architecture for failover cookies.. 51
7. Logging into an e-community. 85
8. e-Community single signon configuration example 90

© Copyright IBM Corp. 2002

Tables

1. Access Manager EPAC fields 5
2. pdwebpi.conf section summary 15
3. Supported Macro Substitutions 17
4. **[proxy]** error page configuration parameters. 19
5. Web server specific configuration parameters 22
6. Authentication audit record field definitions. 24
7. Auditing configuration parameter definitions 26
8. Local Built-in Authenticators 43
9. External CDAS Server Parameters 43
10. BA shared library authentication mechanisms 46
11. Forms shared library authentication mechanism 47
12. Certificate shared library authentication mechanisms. 49
13. Token shared library authentication mechanisms 50
14. IV header field descriptions 53
15. IV Header shared library authentication mechanisms 55
16. HTTP header shared library authentication mechanisms 56
17. IP address shared library authentication mechanisms 57
18. Valid session data types for MPA 59
19. Valid MPA authentication types 60
20. Plug-in ACL permissions 64
21. Plug-in WebDAV permissions 64
22. pdadmin LDAP logon policy commands 65
23. pdadmin LDAP password strength commands 67
24. Password examples 68
25. QOP level descriptions. 74
26. IV header field descriptions 78
27. LTPA configuration parameters 79
28. IV header field descriptions 80
29. General configuration parameters 93
30. Authentication configuration parameters 95
31. Sessions configuration parameters 98
32. LDAP configuration parameters. 98
33. Proxy configuration parameters 99
34. Authorization API configuration parameters 99
35. Web server specific configuration parameters 100
36. Plug-in authentication method / module reference 103
37. Plug-in session module reference 104
38. Plug-in post-authorization module reference 105
39. Plug-in command reference 107

© Copyright IBM Corp. 2002

Preface

IBM® Tivoli® Access Manager Plug-in for Web Servers manages the security of your Web-based resources by acting as the gateway between your clients and secure Web space. The plug-in implements the security policies that protect your Web object space. The plug-in can provide single signon solutions, support web servers running as virtual hosts and incorporate Web application server resources into its security policy.

The *IBM Tivoli Access Manager Plug-in for Web Servers User's Guide* provides installation instructions, administration procedures and technical reference information for securing your Web domain using the Plug-in for Web Servers application.

Who should read this book

This guide is for system administrators responsible for the installation, deployment and administration of Access Manager Plug-in for Web Servers.

Readers should be familiar with the following:

- PC and UNIX® operating systems.
- Database architecture and concepts
- Security management
- Internet protocols, including HTTP, HTTPS and TCP/IP
- Lightweight Directory Access Protocol (LDAP) and directory services
- A supported user registry
- Authentication and authorization

If you are enabling Secure Sockets Layer (SSL) communication, you also should be familiar with SSL protocol, key exchange (public and private), digital signatures, cryptographic algorithms, and certificate authorities.

What this book contains

This book contains the following sections:

- Chapter 1, "Introducing IBM Tivoli Access Manager Plug-in for Web Servers"

  Provides an introduction to the Access Manager Plug-in for Web Servers application giving details of system architecture, functionality and operating environment.
- Chapter 2, "Installing IBM Tivoli Access Manager Plug-in for Web Servers"

  Installation instructions for Access Manager Plug-in for Web Servers including system requirements information and removal procedures.
- Chapter 3, "IBM Tivoli Access Manager Plug-in for Web Servers configuration"

  Provides information on the configuration requirements for Access Manager Plug-in for Web Servers.
- Chapter 4, "IBM Tivoli Access Manager Plug-in for Web Servers authentication"

  Information and configuration instructions for maintaining session state, authenticating requests and supporting post-authorization processing.
- Chapter 5, "IBM Tivoli Access Manager Plug-in for Web Servers security policy"

© Copyright IBM Corp. 2002

Information on the configuration and customization of the Access Manager plug-in for Web Servers security policy.

- Chapter 6, “Web single signon solutions”

  Discussion of single signon solutions for the Web space protected by Access Manager Plug-in for Web Servers.
- Chapter 7, “e-Community single signon”

  Discussion of the Access Manager Plug-in for Web Servers’ e-community single signon solution.
- Appendix A, “pdwebpi.conf reference”

  A listing of the Access Manager Plug-in for Web Servers configuration parameters with associated descriptions.
- Appendix B, “Authentication methods quick reference”

  A listing of all plug-in authentication, session and post-authorization methods with associated descriptions.
- Appendix C, “Command quick reference”

  A listing of the available plug-in utilities with a description of the actions they perform.

Publications

This section lists publications in the IBM Tivoli Access Manager library and any other related documents. It also describes how to access Tivoli publications online, how to order Tivoli publications, and how to make comments on Tivoli publications.

IBM Tivoli Access Manager

The Access Manager library is organized by the following categories:

- Release information
- Base information
- WebSEAL information
- Web Security information
- Developer reference information
- Supplemental technical information

Publications in the product library are included in Portable Document Format (PDF) on the product CD. To access these publications using a Web browser, open the infocenter.html file, which is located in the /doc directory on the product CD.

For additional sources of information about Access Manager and related topics, see the following Web sites:

http://www.ibm.com/redbooks
https://www.tivoli.com/secure/support/documents/fieldguides

Release information

- *IBM Tivoli Access Manager for e-business Read Me First*
  GI11-0918 (`am39_readme.pdf`)

  Provides information for installing and getting started using Access Manager.
- *IBM Tivoli Access Manager for e-business Release Notes*
  GI11-0919 (`am39_relnotes.pdf`)

Provides late-breaking information, such as software limitations, workarounds, and documentation updates.

Base information

- *IBM Tivoli Access Manager Base Installation Guide*
  GC32-0844 (am39_install.pdf)

  Explains how to install, configure, and upgrade Access Manager software, including the Web portal manager interface.
- *IBM Tivoli Access Manager Base Administrator's Guide*
  GC23-4684 (am39_admin.pdf)

  Describes the concepts and procedures for using Access Manager services. Provides instructions for performing tasks from the Web portal manager interface and by using the **pdadmin** command.
- *IBM Tivoli Access Manager Base for Linux on zSeries™ Installation Guide*
  GC23-4796 (am39_zinstall.pdf)

  Explains how to install and configure Access Manager Base for Linux on the zSeries platform.

WebSEAL information

- *IBM Tivoli Access Manager WebSEAL Installation Guide*
  GC32-0848 (amweb39_install.pdf)

  Provides installation, configuration, and removal instructions for the WebSEAL server and the WebSEAL application development kit.
- *IBM Tivoli Access Manager WebSEAL Administrator's Guide*
  GC23-4682 (amweb39_admin.pdf)

  Provides background material, administrative procedures, and technical reference information for using WebSEAL to manage the resources of your secure Web domain.
- *IBM Tivoli Access Manager WebSEAL Developer's Reference*
  GC23-4683 (amweb39_devref.pdf)

  Provides administration and programming information for the Cross-domain Authentication Service (CDAS), the Cross-domain Mapping Framework (CDMF), and the Password Strength Module.
- *IBM Tivoli Access Manager WebSEAL for Linux on zSeries Installation Guide*
  GC23-4796 (amweb39_zinstall.pdf)

  Provides installation, configuration, and removal instructions for WebSEAL server and the WebSEAL application development kit for Linux on the zSeries platform

Web security information

- *IBM Tivoli Access Manager for WebSphere Application Server User's Guide*
  GC32-0850 (amwas39_user.pdf)

  Provides installation, removal, and administration instructions for Access Manager for IBM WebSphere® Application Server.
- *IBM Tivoli Access Manager for WebLogic Server User's Guide*
  GC32-0851 (amwls39_user.pdf)

  Provides installation, removal, and administration instructions for Access Manager for BEA WebLogic Server.
- *IBM Tivoli Access Manager Plug-in for Edge Server User's Guide*
  GC23-4685 (amedge39_user.pdf)

  Describes how to install, configure, and administer the plug-in for IBM WebSphere Edge Server application.

- *IBM Tivoli Access Manager Plug-in for Web Servers User's Guide*
  GC23-4686 (amws39_user.pdf)

  Provides installation instructions, administration procedures, and technical reference information for securing your Web domain using the plug-in for Web servers.

Developer references

- *IBM Tivoli Access Manager Authorization C API Developer's Reference*
  GC32-0849 (am39_authC_devref.pdf)

  Provides reference material that describes how to use the Access Manager authorization C API and the Access Manager service plug-in interface to add Access Manager security to applications.
- *IBM Tivoli Access Manager Authorization Java Classes Developer's Reference*
  GC23-4688 (am39_authJ_devref.pdf)

  Provides reference information for using the Java™ language implementation of the authorization API to enable an application to use Access Manager security.
- *IBM Tivoli Access Manager Administration C API Developer's Reference*
  GC32-0843 (am39_adminC_devref.pdf)

  Provides reference information about using the administration API to enable an application to perform Access Manager administration tasks. This document describes the C implementation of the administration API.
- *IBM Tivoli Access Manager Administration Java Classes Developer's Reference*
  SC32-0842 (am39_adminJ_devref.pdf)

  Provides reference information for using the Java language implementation of the administration API to enable an application to perform Access Manager administration tasks.
- *IBM Tivoli Access Manager WebSEAL Developer's Reference*
  GC23-4683 (amweb39_devref.pdf)

  Provides administration and programming information for the Cross-domain Authentication Service (CDAS), the Cross-domain Mapping Framework (CDMF), and the Password Strength Module.

Technical supplements

- *IBM Tivoli Access Manager Performance Tuning Guide*
  GC43-0846 (am39_perftune.pdf)

  Provides performance tuning information for an environment consisting of Access Manager with IBM SecureWay Directory defined as the user registry.
- *IBM Tivoli Access Manager Capacity Planning Guide*
  GC32-0847 (am39_capplan.pdf)

  Assists planners in determining the number of WebSEAL, LDAP, and backend Web servers needed to achieve a required workload.
- *IBM Tivoli Access Manager Error Message Reference*
  SC32-0845 (am39_error_ref.pdf)

  Provides explanations and recommended actions for the messages produced by Access Manager.

The *Tivoli Glossary* includes definitions for many of the technical terms related to Tivoli software. The *Tivoli Glossary* is available, in English only, at the following Web site:

http://www.tivoli.com/support/documents/glossary/termsm03.htm

Related publications

This section lists publications related to the IBM Tivoli Access Manager library.

IBM DB2® Universal Database

IBM DB2 is required when installing IBM SecureWay® Directory, z/OS®, and OS/390® LDAP servers. DB2 information is available at the following Web site:

http://www-4.ibm.com/software/data/db2/

IBM Global Security Toolkit

Access Manager provides data encryption through the use of IBM Global Security Toolkit (GSKit). GSKit is shipped on the IBM Tivoli Access Manager Base CD for your particular platform.

The GSKit package installs the iKeyman key management utility (gsk5ikm), which enables you to create key databases, public-private key pairs, and certificate requests. The following document is available in the doc/GSKit directory:

- *Secure Sockets Layer Introduction and iKeyman User's Guide*(gskikm5c.pdf)

  Provides information for network or system security administrators who plan to enable SSL communication in their Access Manager secure domain.

IBM SecureWay Directory

IBM SecureWay Directory, Version 3.2.2, is shipped on the IBM Tivoli Access Manager Base CD for your particular platform. If you plan to install the IBM SecureWay Directory server as your user registry, the following documents are available in the `/doc/Directory` path on the IBM Tivoli Access Manager Base CD for your particular platform:

- *IBM SecureWay Directory Installation and Configuration Guide*, (`aparent.pdf, lparent.pdf, sparent.pdf, wparent.pdf`)

  Provides installation, configuration, and migration information for IBM SecureWay Directory components on AIX®, Linux, Solaris Operating Environment, and Microsoft® Windows® operating systems.

- *IBM SecureWay Directory Release Notes* (`relnote.pdf`)

  Supplements IBM SecureWay Directory, Version 3.2.2, product documentation and describes features and functions made available to you in this release.

- *IBM SecureWay Directory Readme Addendum* (`addendum322.pdf`)

  Provides information about changes and fixes that occurred after the IBM SecureWay Directory documentation had been translated. This file is in English only.

- *IBM SecureWay Directory Server Readme* (`server.pdf`)

  Provides a description of the IBM SecureWay Directory Server, Version 3.2.2.

- *IBM SecureWay Directory Client Readme* (`client.pdf`)

  Provides a description of the IBM SecureWay Directory Client SDK, Version 3.2.2. This software development kit (SDK) provides LDAP application development support.

- *SSL Introduction and iKeyman User's Guide* (`gskikm5c.pdf`)

  Provides information for network or system security administrators who plan to enable SSL communication in their Access Manager secure domain.

- *IBM SecureWay Directory Configuration Schema*
  (scparent.pdf)

  Describes the directory information tree (DIT) and the attributes that are used to configure the slapd32.conf file. In Version 3.2, the directory settings are stored using the LDAP Directory Interchange Format (LDIF) in the slapd32.conf file.
- *IBM SecureWay Directory Tuning Guide*
  (tuning.pdf)

  Provides performance tuning information for IBM SecureWay Directory. Tuning considerations for directory sizes ranging from a few thousand entries to millions of entries are given where applicable.

For more information about IBM SecureWay Directory, see the following Web site:

http://www.software.ibm.com/network/directory/library/

IBM WebSphere Application Server

IBM WebSphere Application Server, Advanced Single Server Edition 4.0.2, is installed with the Web portal manager interface. For information about IBM WebSphere Application Server, see the following Web site:

http://www-4.ibm.com/software/webservers/appserv/infocenter.html

Accessing publications online

Publications in the product libraries are included in Portable Document Format (PDF) on the product CD. To access these publications using a Web browser, open the infocenter.html file, which is located in the /doc directory on the product CD.

When IBM publishes an updated version of one or more online or hardcopy publications, they are posted to the Tivoli Information Center. The Tivoli Information Center contains the most recent version of the publications in the product library in PDF or HTML format, or both. Translated documents are also available for some products.

You can access updated publications in the Tivoli Information Center from the following Tivoli Customer Support Web site:
http://www.tivoli.com/support/documents/

Information is organized by product, including release notes, installation guides, user's guides, administrator's guides, and developer's references.

Note: If you print PDF documents on other than letter-sized paper, select the **Fit to page** check box in the Adobe Acrobat Print dialog (which is available when you click **File → Print**) to ensure that the full dimensions of a letter-sized page are printed on the paper that you are using.

Ordering publications

You can order many Tivoli publications online at the following Web site:

http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi

You can also order by telephone by calling one of these numbers:

- In the United States: 800-879-2755
- In Canada: 800-426-4968

- In other countries, for a list of telephone numbers, see the following Web site:
  http://www.tivoli.com/inside/store/lit_order.html

Providing feedback about publications

We are very interested in hearing about your experience with Tivoli products and documentation, and we welcome your suggestions for improvements. If you have comments or suggestions about our products and documentation, contact us in one of the following ways:

- Send an e-mail to pubs@tivoli.com.
- Complete our customer feedback survey at the following Web site:
  http://www.tivoli.com/support/survey/

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate all features of the graphical user interface.

Contacting customer support

If you have a problem with any Tivoli product, you can contact Tivoli Customer Support. See the *Tivoli Customer Support Handbook* at the following Web site:

http://www.tivoli.com/support/handbook/

The handbook provides information about how to contact Tivoli Customer Support, depending on the severity of your problem, and the following information:

- Registration and eligibility
- Telephone numbers and e-mail addresses, depending on the country in which you are located
- What information you should gather before contacting support

Conventions used in this book

This guide uses several conventions for special terms and actions, operating system-dependent commands and paths, and margin graphics.

Typeface conventions

The following typeface conventions are used in this book:

Bold	Command names and options, keywords, and other information that you must use literally appear in **bold**.
Italic	Variables, command options, and values you must provide appear in italics. Titles of publications and special words or phrases that are emphasized also appear in *italics*.
`Monospace`	Code examples, command lines, screen output, file and directory names, and system messages appear in `monospace` font.

Chapter 1. Introducing IBM Tivoli Access Manager Plug-in for Web Servers

IBM Tivoli Access Manager (Access Manager) Plug-in for Web Servers is an integration solution that facilitates the implementation and management of security policy for your protected web space. Installed as part of the same process as your web server, the plug-in acts as the security gateway between your clients and your protected web space.

This introductory chapter gives an overview of Access Manager Plug-in for Web Servers technology, it identifies the technical requirements for the product and provides an introduction to the processes for ensuring the security of your web space using the plug-in.

Topic Index:

- "Understanding Access Manager Plug-in for Web Servers technology"
- "Protecting your Web space with Access Manager Plug-in for Web Servers" on page 3
- "Understanding Access Manager Plug-in for Web Servers authentication" on page 4
- "Understanding credential acquisition" on page 5

Understanding Access Manager Plug-in for Web Servers technology

Access Manager Plug-in for Web Servers operates as part of the same process as your Web server, intercepting each request that arrives, determining if an authorization decision is required and providing a means for user authentication if necessary.

Access Manager Plug-in for Web Servers integrates with the IBM Tivoli Access Manager application to provide a complete security solution for your Web resources. The plug-in can provide single signon solutions and incorporate Web application resources into its security policy.

Basic operational components and architecture

Two basic architectural components make up Access Manager Plug-in for Web Servers – the plug-in component and the Authorization Server. The plug-in component operates with Web server threads sending details of each request, via an Inter-Process Communication (IPC) interface, to the Authorization Server. The Authorization Server performs the authentication and authorization of incoming requests. The Authorization Server is a local mode AZNAPI application that accepts and processes requests from the plug-in and responds, telling the plug-in

© Copyright IBM Corp. 2002

how to handle each request.

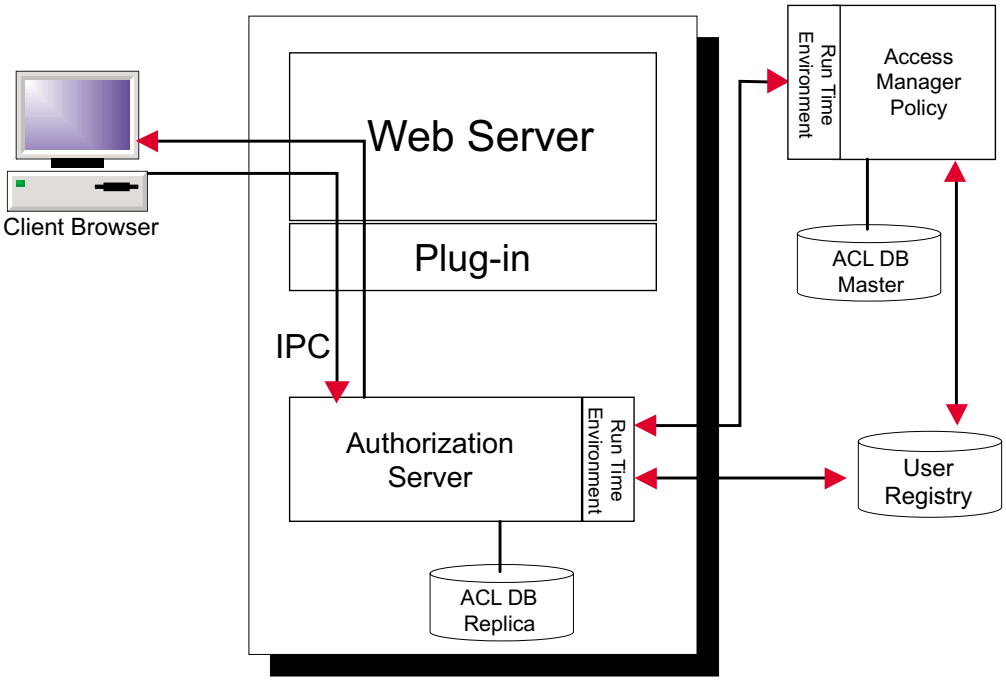

Figure 1. Plug-in and Access Manager component interaction.

The Authorization server determines which virtual host the request is addressed to (if virtual hosts are present on the Web server) and determines if the request requires authorization. For requests that do not require authorization, it allows the Web server to handle the request. The Authorization server performs the following actions for requests that require authorization:

1. Extracts authentication or session information if the request has been previously authenticated.
2. Initiates an authentication interaction with the user if necessary.
3. Constructs an Access Manager credential.
4. Identifies the resources the user will access and maps these resources to the corresponding Access Manager protected object name. A protected object name represents an electronic entity such as a secure part of a web site or an application that only certain users are permitted to access.
5. Determines if the request or response requires modification
6. Generates any response required by the plug-in or the host Web server by adding cookies or headers to the request/response or by generating a response (e.g. an authenticated response or an unauthorized response)

Support for virtual hosts

Virtual Hosting is the capability of a web server that allows it to appear as more than one host to the Internet. The web servers supported by Access Manager Plug-in for Web Servers all provide virtual hosting capability.

Access Manager Plug-in for Web Servers provides the capability to implement security policy on a per virtual host basis. The application setup required to implement this ability is the subject of later sections in this document.

Protecting your Web space with Access Manager Plug-in for Web Servers

Access Manager Plug-in for Web Servers provides the following features:

- Supports multiple authentication methods, including: basic authentication, IP address, token, certificates, and forms, among others.
- Accepts HTTP and HTTPS requests
- Protects Web server resources by authenticating and authorizing user requests dependant upon organizational policy.
- Supports the authentication and authorization of requests in a virtual host environment.
- Manages access control to the Web server space.

  Supported resources include URLs, URL-based regular expressions, CGI programs, HTML files, Java servlets, and Java class files.
- Caches session and credential information to eliminate repetitive queries to the user registry database during authorization checks.
- Provides single signon capabilities

Planning and implementing security policy

A corporate security policy identifies the Web resources requiring protection and the level of protection required for each of those web resources. Access Manager uses a virtual representation of these Web resources, called the protected object space. The protected object space contains objects that represent actual physical resources in your network. You implement security policy by applying the appropriate security mechanisms to the objects requiring protection.

Security mechanisms include:

- Access control list (ACL) policies

  ACL policies identify user types who can be considered for access and specify the operations permitted on the object for each user type.
- Protected object policies (POP)

  A POP specifies additional conditions governing the access to the protected object, such as privacy, integrity, auditing, and time-of-day access.
- Extended attributes

  Extended attributes are additional values placed on an object, ACL, or POP that can influence an authorization decision.

It is the Authorization Server component of Access Manager Plug-in for Web Servers that permits or denies access to protected resources based on the user's credentials and the access controls placed on the objects. To successfully implement security policy, you must logically organize the different content types and apply the appropriate ACL and POP policies. Access control management can be complex and is made easier by careful categorization of the content types. Comprehensive information on Access Manager including details of setting policy can be found in the *IBM Tivoli Access Manager Base Administrator's Guide*.

Understanding Access Manager Plug-in for Web Servers authentication

Authentication is the method of identifying an individual process or entity attempting to log on to a secure domain. Authorization is the method of determining whether an authenticated user has the right to perform an operation on a specific resource. Authentication ensures that the individual is who they claim to be, but says nothing about the ability to perform operations on a resource.

Access Manager Plug-in for Web Servers enforces a high degree of security in a secure domain by requiring each client to provide proof of identity. Comprehensive network security can be provided by having Access Manager Plug-in for Web Servers control the authentication and authorization of clients.

The following conditions apply to Access Manager Plug-in for Web Servers authentication:

- The plug-in supports a standard set of authentication methods. You can customize the plug-in to support other authentication methods.
- The plug-in process is independent of the authentication method.
- The plug-in only requires a client identity. From this identity, the plug-in obtains an authenticated (or unauthenticated) credential that can be used by the Authorization Server to permit or deny access to resources.

This flexible approach to authentication allows security policy to be based on business requirements and not physical network topology.

The goals of authentication

The Access Manager Plug-in for Web Servers authentication process results in the following actions:

1. Client authentication results in a client identity.

   Client authentication is only successful if the user has an account defined in the Access Manager user registry. Otherwise the user is designated as unauthenticated.

2. Access Manager Plug-in for Web Servers uses the client identity to acquire credentials for that client.

   The plug-in matches the authenticated client identity with a registered Access Manager user. The plug-in then obtains the appropriate user credentials appropriate. This is known as credentials acquisition.

   Credentials include the user name and any groups where the user has membership. These credentials are available to the plug-in which permits or denies access to requested objects in the Access Manager protected object space.

   Credentials can be used by any Access Manager service that requires information about the client. Credentials allow Access Manager to securely perform a multitude of services such as authorization, auditing, and delegation.

   See Chapter 4, “IBM Tivoli Access Manager Plug-in for Web Servers authentication” on page 29 for further information about support for specific authentication methods.

Understanding credential acquisition

The primary goal of the authentication process is to acquire credential information describing the client user. The user credential is a key requirements for participating in the secure domain.

Access Manager distinguishes the authentication of the user from the acquisition of credentials. A user's identity is always constant. However, credentials – which define the groups or roles in which a user participates – are variable. Context-specific credentials can change over time. For example, when a person is promoted, credentials must reflect the new responsibility level.

The authentication process results in method-specific user identity information. This information is checked against user account information that resides in the Access Manager user registry (LDAP by default). Access Manager Plug-in for Web Servers maps the user name and group information to a common domain-wide representation and format known as the Extended Privilege Attribute Certificate (EPAC).

Method-specific identity information, such as passwords, tokens, and certificates, represent physical identity properties of the user. This information can be used to establish a secure session with the server.

The resulting credential, which represents a user's privileges in the secure domain, describes the user in a specific context and is only valid for the lifetime of that session.

Access Manager credentials contain the user identity and groups where this user has membership.

The Extended Privilege Attribute Certificate (EPAC)

Credentials are used by any Access Manager service that requires information about the client. For example, the Access Manager Authorization Server uses credentials to determine whether a user is authorized to perform specific operations on a protected resource in the secure domain. Credentials are also used in other tasks such as logging and auditing.

EPACs contain the Unique Universal Identifiers (UUIDs) that Access Manager needs to work with access control lists (ACLs).

The following EPAC fields are appropriate to Access Manager:

Table 1. Access Manager EPAC fields

Attribute	Description
Secure Domain ID	Principal's home secure domain identifier
Principal UUID	UUID of the principal
Group UUIDs	UUID(s) of groups to which the principal belongs

Chapter 2. Installing IBM Tivoli Access Manager Plug-in for Web Servers

This chapter provides details of the IBM Tivoli Access Manager (Access Manager) Plug-in for Web Servers installation. Included is information on the hardware and software requirements as well as detailed installation instructions.

Topic index:

- "Supported platforms"
- "Disk and memory requirements"
- "Software prerequisites"
- "Installing Access Manager Plug-in for Web Servers" on page 8
- "Removing Access Manager Plug-in for Web Servers" on page 12

Supported platforms

Access Manager Plug-in for Web Servers integrates with the following web servers on the following platforms:

- Windows 2000 Server / Advanced Server – Service pack 2 with Internet Information Server (IIS) version 5.0.
- Solaris Operating Environment 7 (sparc) with iPlanet 6.0
- AIX 5L with the IBM HTTP Server (IHS) 1.3.19

Note: Tivoli recommends that all security fixes from Web server vendors are applied.

Disk and memory requirements

Access Manager Plug-in for Web Servers has the following hardware requirements:

- When combined with the prerequisite Access Manager runtime environment the minimum required disk space is 23MB
- Memory: 64 MB minimum. 256 MB recommended.

  Note that the 64 MB minimum is in addition to the 64 MB minimum needed by the prerequisite Access Manager runtime environment. A total memory of 256 MB or greater will produce optimum performance results.

Software prerequisites

Access Manager Plug-in for Web Servers is an application that integrates with web server software and runs in an Access Manager secure domain. Before installing the plug-in you must set up your web server and create an Access Manager secure domain.

The Access Manager secure domain is established when you install the Access Manager software. This software is distributed on the IBM Tivoli Access Manager for e-business Base CD.

The following software must be installed on the target Web server prior to installing Access Manager Plug-in for Web Servers software:

© Copyright IBM Corp. 2002

- The Web server software. This will be either:
  - IIS 5.0 for a Windows 2000 Server / Advanced Server environment
  - iPlanet 6.0 for Solaris Operating Environment 7 (sparc)
  - IHS 1.3.19 for an AIX 5L environment.
- IBM Tivoli Access Manager runtime environment v3.9

The following applications do not need to be installed on the Web server – they are setup as part of establishing the Access Manager secure domain. They must exist somewhere within the network accessible by the plug-in.

- IBM Tivoli Access Manager Policy Server v3.9
- IBM Global Security Toolkit (GSKit) 5.0.4.65
- If using LDAP, a supported LDAP server, such as IBM Secure Way Directory 3.2.2

Installing Access Manager Plug-in for Web Servers

This section provides the instructions for installing Access Manager Plug-in for Web Servers on the three supported platforms.

Installing the plug-in on AIX – IHS

To install and configure Access Manager Plug-in for Web Servers on AIX:

1. On your AIX 5L Web server, ensure you have available in your environment:
   - IBM Tivoli Access Manager for e-business Policy Server v3.9. Refer to the *IBM Tivoli Access Manager Base Installation Guide.*

     Note: The Access Manager Policy Server does not need to reside on the same machine as the Access Manager runtime environment.
2. Ensure that you have installed:
   - IHS Web server software.
   - IBM Tivoli Access Manager for e-business runtime environment v3.9. Refer to the *IBM Tivoli Access Manager Base Installation Guide.*
   - IBM Global Security Toolkit (GSKit) 5.0.4.65
   - If you are using an LDAP user registry, install the IBM SecureWay Directory Client 3.2.2
3. The Access Manager Plug-in for Web Servers installation separates the extraction from the package configuration. Use **SMIT** to install software packages on AIX. Then use the plug-in configuration utility **pdwpicfg** to configure the installation.

   Log on as user **root**.
4. Insert the *IBM Tivoli Access Manager Web Security, Version 3.9, for AIX* CD into the CD drive.
5. Enter the following command at a shell prompt:

   ```
   # smit
   ```

   The **SMIT** utility starts.
6. Select **Software Installation and Maintenance**. Select **Install and Update Software**. Select **Install and Update Software from LATEST Available Software**.
7. When prompted for input device, enter the location where the CD is mounted.
8. Click the **List** button for **SOFTWARE to install**.

A multi-select list window displays the IBM Tivoli Access Manager software packages.

9. Select the **Access Manager Plug-in for Web Servers** software package. Click **OK**.
10. The **Install and Update Software from LATEST Available Software** dialog is displayed.
11. Verify that the default value of **yes** is present in the field labeled **AUTOMATICALLY install requisite software**.
12. Set other fields to values appropriate to your installation. In most cases, you can accept the default values. Click **OK**.
13. A message box is displayed asking if you are sure you want to install this package. Click **OK**.

    The package files are installed. Several status messages are displayed. A final status message indicates success upon completion of the file extraction.
14. Repeat steps 8 – 12 for the **Access Manager Plug-in for IBM HTTP Server** package.
15. Click **Done**. Click **Cancel** to exit SMIT.
16. If you have not already configured the Access Manager runtime environment, it must be done at this stage. Refer to the *IBM Tivoli Access Manager Base Installation Guide* for details on configuring the Access Manager runtime environment.
17. To configure the plug-in, move to `/opt/pdwebpi/bin` and run:

    ```
    # ./pdwpicfg
    ```

    Enter the letter `c`.
18. A list of all the virtual hosts known to the Web server is displayed. You have three options:
    - If you only want one virtual host protected by the plug-in, enter the number relating to the virtual host in the displayed list.
    - To secure more than one virtual host, enter values relating to the positions of the virtual hosts in the displayed list. Separate the entered numbers by spaces.
    - Enter `all` to have the plug-in protect all the known virtual hosts on the server.
19. Enter the Access Manager Administrator ID and password.
20. An AZN update is the transfer of policy information delta packets from the authorization policy server during application operation. Enter the port number to listen for AZN updates, or press **Return** to accept the default.
21. Enter `Y/N` to enable/disable SSL communication to the LDAP server. Enabling SSL is probably not necessary in environments where the Web server and LDAP server reside in the same secure network. If you can be certain of the integrity of data sent between the Web server and LDAP, choosing not to use SSL will improve network bandwidth by removing the security overhead.
22. If you enabled SSL communications between the plug-in and LDAP server, you are prompted to enter the LDAP SSL client key file.
23. The Access Manager Plug-in for Web Servers configuration should complete successfully.

Installing the plug-in on the Solaris Operating Environment – iPlanet

To install and configure Access Manager Plug-in for Web Servers on the Solaris Operating Environment:

1. On your Solaris Operating Environment Web server, ensure you have available in your environment:
   - IBM Tivoli Access Manager for e-business Policy Server v3.9. Refer to the *IBM Tivoli Access Manager Base Installation Guide.*

     Note: The Access Manager Policy Server does not need to reside on the same machine as the Access Manager runtime environment.
2. Ensure that you have installed:
   - iPlanet Web server software.
   - IBM Tivoli Access Manager for e-business runtime environment v3.9. Refer to the *IBM Tivoli Access Manager Base Installation Guide.*
   - IBM Global Security Toolkit (GSKit) 5.0.4.65
   - If you are using an LDAP user registry, install the IBM SecureWay Directory Client 3.2.2
3. The plug-in installation separates file extraction from package configuration. Use **pkgadd** to install software packages on the Solaris Operating Environment. Then use the plug-in configuration utility **pdwpicfg** to configure the plug-in.

   Log on as user **root**.
4. Mount the *IBM Tivoli Access Manager Web Security, Version 3.9, for Solaris CD* on `/cdrom/cdrom0`
5. Change directory to `/cdrom/cdrom0/solaris`
6. The plug-in installation requires the adding of two packages. Perform the following command to install the plug-in:

   ```
   # pkgadd -d . PDWPI PDWPIipl
   ```

   When prompted enter `y` and press **Return**. Files are extracted from the CD and installed on the hard disk.
7. To configure the plug-in, move to `/opt/pdwebpi/bin` and run:

   ```
   # ./pdwpicfg
   ```
8. Enter the letter `c` to configure the application.
9. Enter the root directory of the iPlanet server.
10. A list of all the virtual hosts known to the Web server is displayed. You have three options:
    - If you only want one virtual host protected by the plug-in, enter the number relating to the virtual host in the displayed list.
    - To secure more than one virtual host, enter values relating to the positions of the virtual hosts in the displayed list, separating the entered numbers by spaces.
    - Enter `all` to have the plug-in protect all the known virtual hosts on the server.
11. Enter the Access Manager Administrator ID and password.
12. An AZN update is the transfer of policy information delta packets from the authorization policy server during application operation. Enter the port number to listen for AZN updates, or press **Return** to accept the default.

13. Enter `Y/N` to enable/disable SSL communication to the LDAP server. Enabling SSL is probably not necessary in environments where the Web server and LDAP server reside in the same secure network. If you can be certain of the integrity of data sent between the Web server and LDAP, choosing not to use SSL will improve network bandwidth by removing the security overhead.
14. If you enabled SSL communications between the plug-in and LDAP server, you are prompted to enter the LDAP SSL client key file.
15. The Access Manager Plug-in for Web Servers configuration should complete successfully.

Installing the plug-in on Windows – IIS

To install Access Manager Plug-in for Web Servers on your Windows 2000 Server/Advanced Server Web server:

1. On your Windows 2000 Web server, ensure you have available in your environment:
   - IBM Tivoli Access Manager for e-business Policy Server v3.9. Refer to the *IBM Tivoli Access Manager Base Installation Guide.*

     Note: The Access Manager Policy Server does not need to reside on the same machine as the Access Manager runtime environment.
2. Ensure that you have installed:
   - IIS Web server software.
   - IBM Tivoli Access Manager for e-business runtime environment v3.9. Refer to the *IBM Tivoli Access Manager Base Installation Guide.*
   - IBM Global Security Toolkit (GSKit) 5.0.4.65
   - If you are using an LDAP user registry, install the IBM SecureWay Directory Client 3.2.2
3. Log on to the Windows domain as a user with Windows administrator privileges.
4. Insert the *IBM Tivoli Access Manager Web Security, Version 3.9, for Windows* CD into the CD drive.
5. Run the Access Manager Plug-in for Web Servers InstallShield setup program by double-clicking on the following file (where the letter E: is the CD drive).

   `E:\Windows\PolicyDirector\Disk Images\Disk1\setup.exe`
6. From the **Select Packages** window, select the Plug-in for Web Servers package and click **OK**.
7. The **Choose Setup Language** dialog is displayed. Select the appropriate language and click **OK**
8. The **InstallShield** program starts and the **Welcome** dialog box is displayed. Click **Next**.
9. The **License Agreement** dialog is displayed. Click **Yes** to accept the conditions of the license agreement.
10. The **Select Packages** dialog is displayed. Keep both the options, **Access Manager Plug-in for Web Servers** and **Access Manager Plug-in for Microsoft Internet Information Services** checked. Click **Next**.
11. The **Choose Destination Location** dialog is displayed. Accept the default installation location or specify an alternative location. Click **Next**.

    The program files are extracted to disk. A message is displayed indicating that the software has been installed.
12. Click **Finish** to exit the setup program.

13. From the **Start** menu select: **Programs** > **Access Manager Plug-in for Web Servers** > **Configuration**

    The Access Manager Plug-in for Web Servers configuration selection dialog is displayed.
14. A list of all the virtual hosts known to the Web server is displayed. Select the virtual hosts you want to protect. Click **Next**.
15. Enter the Access Manager administrator user ID and password. Click **Next**.
16. An AZN update is the transfer of policy information delta packets from the authorization policy server during application operation. Enter the port number to listen for AZN updates, or accept the default. Click **Next**.
17. Select **Yes** or **No** to enable/disable SSL communication to the LDAP server. Enabling SSL is probably not necessary in environments where the Web server and LDAP server reside in the same secure network. If you can be certain of the integrity of data sent between the Web server and LDAP, choosing not to use SSL will improve network bandwidth by removing the security overhead.

    If you choose to use SSL communications between the plug-in and LDAP server:

    a. Enter the path and filename of the key file used to encrypt SSL.
    b. If required, enter the certificate label.
    c. Enter the key file password.

    Select **Next**.
18. The Access Manager Plug-in for Web Servers configuration should complete successfully.
19. Restart IIS.

Removing Access Manager Plug-in for Web Servers

This section describes the process for removing Access Manager Plug-in for Web Servers. This section does not describe the process for removing the Access Manager runtime environment or the Access Manager Policy Server. For details on removing the runtime environment and Policy Server refer to the *IBM Tivoli Access Manager Base Installation Guide*.

Removing the plug-in from Windows – IIS

The plug-in must be unconfigured before it is removed.

To unconfigure the plug-in on Windows:

1. Log on as a Windows user with administrative privileges.
2. From the **Start** menu click: **Programs** > **Access Manager Plug-in for Web Servers** > **Unconfiguration**

   Note: If run from the command prompt, the **-f** option can be used to force unconfiguration when the Management Server cannot be reached.
3. A list of all the virtual hosts protected by the plug-in is displayed. Select the virtual hosts you want to unconfigure. Click **Next**.
4. Enter the Access Manager user ID and Password. Select **Next**.

   A status message is displayed once the plug-in is successfully unconfigured.

To remove the plug-in from Windows:

1. From Windows Control Panel, click **Add/Remove Programs**.

The **Add/Remove Programs** dialog is displayed, listing all installed software.

2. Select the entry for **Access Manager Plug-in for Microsoft Internet Information Services**. Click the **change/remove** button.
3. The **InstallShield** program starts and removes the plug-in.
4. Click **Finish**.

Removing the plug-in from AIX – IHS

Before removal the plug-in needs to be unconfigured. To unconfigure the plug-in on an AIX platform:

1. Log on as root
2. Start the plug-in configuration utility from the bin directory by running:

   ```
   # pdwpicfg
   ```

   Note: The **-f** option can be used to force unconfiguration when the Management Server cannot be reached.
3. Enter `u` for unconfiguration.
4. A list of the protected virtual hosts is displayed. Select the virtual hosts you want to unconfigure.
5. Enter the Access Manager administrator ID and password.
6. A message is displayed when the unconfiguration is complete.

To remove the plug-in:

1. As user root, start **SMIT**
2. Select **Communications Applications and Services**.
3. The **Communications Applications and Services** menu is displayed. Select **Access Manager**.
4. From the **Access Manager** menu select **Access Manager Unconfiguration**. A list of configured IBM Tivoli Access Manager packages is displayed.
5. Select **Access Manager Plug-in for Web Servers**.

   When prompted enter the Access Manager password.
6. Press **Enter** through any prompts.
7. Repeat steps 3 to 7 for the **Access Manager Plug-in for Web Servers IHS** package.

Removing the plug-in from Solaris Operating Environment – iPlanet

You must unconfigure the plug-in before it can be removed. To unconfigure the plug-in on Solaris Operating Environment:

1. Log on as root
2. Start the plug-in configuration utility from the bin directory by running:

   ```
   # pdwpicfg
   ```

   Note: The **-f** option can be used to force unconfiguration when the Management Server cannot be reached.
3. Enter `u` for unconfiguration.
4. A list of the protected virtual hosts is displayed. Select the virtual hosts you want to unconfigure.
5. Enter the Access Manager administrator ID and password.
6. A message is displayed when the unconfiguration is complete.

To remove the plug-in from the Solaris Operating Environment:

1. Enter the command:

   ```
   # pkgrm PDWPI PDWPIipl
   ```

   You are prompted to confirm your decision. Enter y at the prompts.

   A message is displayed indicating the success of the removal.

Chapter 3. IBM Tivoli Access Manager Plug-in for Web Servers configuration

This chapter describes general administration and configuration tasks you can perform to customize IBM Tivoli Access Manager (Access Manager) Plug-in for Web Servers.

Topic Index:

- "General plug-in information"
- "Configuring the Authorization Server" on page 18
- "Configuring for virtual host servers" on page 19
- "Web server specific configuration" on page 22
- "Configuring plug-in auditing, logging, tracing and the cache database" on page 24
- "Configuring the authorization API service" on page 27

General plug-in information

The following sections describe general information about Access Manager Plug-in for Web Servers:

- "Introducing the pdwebpi.conf configuration file"
- "The pdwebpimgr.conf configuration file" on page 16
- "Root directory of the Access Manager Plug-in for Web Servers installation" on page 16
- "Starting and stopping Access Manager Plug-in for Web Servers" on page 16
- "HTTP error messages" on page 17

Introducing the pdwebpi.conf configuration file

You can customize the operation of the plug-in by configuring the parameters located in the `pdwebpi.conf` configuration file. The file is located in the following directory:

UNIX:

```
/opt/pdwebpi/etc/
```

Windows:

```
C:\Program Files\Tivoli\PDWebPI\etc\
```

The following table categorizes the configuration file's stanzas.

Table 2. pdwebpi.conf section summary

Sections	Stanzas
GENERAL	[module-mgr][modules][wpiconfig][pdweb-plugins]
AUTHENTICATION	[common-modules][authentication-levels][authentication-mechanisms][BA] [failover] [forms][ltpa] [tag-value][token-card] [http-hdr] [iv-headers] [acctmgmt][ecsso][ecsso-domain-keys]
VIRTUAL HOSTS	[*virtual-host-name*]

© Copyright IBM Corp. 2002

Table 2. pdwebpi.conf section summary (continued)

Sections	Stanzas
SESSIONS	[sessions] [session-cookie]
LDAP	[ldap]
PROXY	[ipc][proxy]
AUTHORIZATION API	[aznapi-entitlement-services][aznapi-admin-services][aznapi-configuration]
WEB SERVER	[ihs][iis][iis:minimum-post-data][iplanet]

See Appendix A, “pdwebpi.conf reference” on page 93 for a description of the configurable parameters within the `pdwebpi.conf` configuration file.

Note: Anytime you make a change to the `pdwebpi.conf` file, you must manually restart Access Manager Plug-in for Web Servers so that the new changes are recognized. See “Starting and stopping Access Manager Plug-in for Web Servers” for information on starting and stopping the application.

The pdwebpimgr.conf configuration file

UNIX installations of the plug-in include the configuration file `pdwebpimgr.conf`. This configuration file contains parameters that are used to automatically re-start the authorization daemon if it has crashed.

The file is located in the directory:

```
/opt/pdwebpi/etc/
```

There is no reason why you should need to change the parameters in this file.

Root directory of the Access Manager Plug-in for Web Servers installation

The Access Manager Plug-in for Web Server’s program files are installed in the following root directory:

UNIX:

```
/opt/pdwebpi/
```

Windows:

```
C:\Program Files\Tivoli\PDWebPI\
```

You can configure this path during a Windows installation of the plug-in. You cannot configure this path on UNIX installations. This guide uses the *install_path* variable to represent this root directory.

On UNIX installations, the following separate directory contains expandable files, such as audit and log files:

```
/var/pdwebpi/
```

Starting and stopping Access Manager Plug-in for Web Servers

To start and stop the plug-in process, use the **pdwebpi_start** command on UNIX and use the Services Control Panel on Windows.

UNIX:

```
pdwebpi_start {start|stop|restart|status}
```

For example, to stop the plug-in and then restart it, use:

```
# pdwebpi_start restart
```

The **pdwebpi_start** command is located in the following directory:

```
/opt/pdwebpi/sbin/
```

Windows:
Identify the plug-in process in the Services Control Panel and use the appropriate control buttons.

HTTP error messages

Sometimes Access Manager Plug-in for Web Servers attempts to service a request and fails. There can be many causes for this failure. Two common causes of failure are:

- A file does not exist
- Permission settings forbid access

When a failure to service a request occurs, the plug-in returns an error code to the web server which interprets the error code and displays a corresponding error page.

Macro support

The following macros are available for use in customizing HTML error pages. Macros dynamically substitute appropriate information that is available.

Table 3. Supported Macro Substitutions

Macro	Description
%USERNAME%	The name of the logged in user
%ERROR_CODE%	A numeric error code associated with an error
%ERROR_TEXT%	Error text associated with an error
%URL%	The URL requested by the client
%HOSTNAME%	Fully qualified hostname
%HTTP_BASE%	Base HTTP URL of the server: http://*host*:*tcpport*/
%HTTPS_BASE%	Base HTTPS URL of the server: https://*host*:*sslport*/
%REFERER%	The value of the referer header from the request, or 'Unknown' if none
%BACK_URL%	The value of the referer header from the request, or '/' if none
%BACK_NAME%	The value 'BACK' if a referer header is present in the request, or 'HOME' if none.

Configuring the Authorization Server

The majority of authorization and authentication processing is handled by the Authorization Server. The Authorization Server provides a pool of worker threads that:

- Accept requests from the plug-in
- Send results of each request back to the plug-in

The plug-in communicates with the Authorization Server via an IPC mechanism implemented using shared memory. The **[ipc]** stanza in the `pdwebpi.conf` configuration file specifies configuration parameters that pertain to communication between the plug-in and the Authorization Server.

Configuring Worker Threads

The **number-of-workers** and the **worker-size** parameters in the **[ipc]** stanza of the configuration file, specify values that can be tuned to provide optimal plug–in Authorization Server performance. How you set these values will depend on the quantity and type of traffic on your network.

```
[ipc]
number-of-workers = 10
worker-size = 10000
cleanup-interval=300
```

The **number-of-workers** parameter specifies the number of concurrent incoming requests that can be serviced by the plug-in. Requests that arrive when all worker threads are busy are buffered until a worker thread becomes available. This parameter simply specifies the number of threads made available to service a potentially unlimited work queue. It should be increased according to the maximum number of requests you expect the Web server to be accepting simultaneously. Increasing this value on UNIX platforms may require limits.

By increasing the number of threads, you are, in general, decreasing the average time it takes to finish the requests. However, increasing the number of threads impacts other factors which could have an adverse effect on server performance.

The **worker-size** parameter defines the amount of memory (in bytes) that is pre-allocated for each worker thread.

The **cleanup-interval** is the time in minutes between successive clean-ups of the Authorization Server's shared memory.

Note: It is recommended that the **cleanup-interval** and **worker-size** parameters only be changed to troubleshoot performance problems.

Setting the Maximum Session Lifetime

The **max-session-lifetime** parameter in the **[ipc]** stanza of the `pdwebpi.conf` configuration file sets the time in minutes that the plug-in will wait for a response from the Authorization Server before timing out. If such a timeout occurs an error page is sent to the client. Such timeouts are highly unlikely.

```
[ipc]
max-session-lifetime = 300
```

Configuring Error Pages

Located in the **[proxy]** stanza of the `pdwebpi.conf` configuration file are parameters for specifying the HTML pages to be displayed when errors occur in the proxy. The parameters set within the **[proxy]** stanza are: **error-page**, **acct-locked-page** and **retry-limit-reached-page**. Default files exist for these parameters which can be be edited or a new file specified to suit your organization's requirements. The following table summarizes the parameters.

Table 4. **[proxy]** *error page configuration parameters.*

Parameter	Description
error-page	The path to the page displayed on the users browser when an unexpected server error occurs.
acct-locked-page	The path to the page displayed when a user attempts to access a locked account.
retry-limit-reached-page	The path to the page displayed when the allowed maximum number of failed logon attempts has been reached. The maximum number of allowed logon failures is set in LDAP - refer to, "Three strikes logon policy" on page 65 for details on setting this value.

By default, the sample HTML pages are located in the following directory: *`install_directory`*`/nls/html/`*`lang`*.

Where *lang* is taken from the NLS configuration. In a US English installation this will be set to **C**.

Configuring for virtual host servers

Virtual hosts are identified to Access Manager Plug-in for Web Servers by an arbitrary name that is set in the **[pdweb-plugins]** stanza of the `pdwebpi.conf` configuration file.

The plug-in can apply distinct security policy according to two characteristics of a request:

- an ID for the virtual host to which the request was addressed
- the protocol (http or https) over which the request arrived

The virtual host ID is derived from the host Web server's configuration information and is Web server specific. It is determined as follows:

IHS The virtual host ID is derived in the following order of precedence:

1. It is taken from the server name inline in the **<VirtualHost....>** block; i.e., **<VirtualHost** *servername*:*port*>
2. If a server name does not exist as above, the virtual host ID is taken from the **ServerName** directive in the **<VirtualHost....>** block; i.e., **Servername** *servername*
3. If a server name still does not exist, the virtual host ID is taken from the global **ServerName** directive outside of the **<VirtualHost....>** block; i.e., **ServerName** *servername*
4. If still no server name exists, the virtual host ID is taken from the fully_qualified_domain_name(gethostname())

If the port on which the virtual host listens (usual inline in the **<VirtualHost** *servername*:*port*>) is not either 80 or 443, the port number is appended to the virtual host ID (i.e. a port of 8080 would lead to a virtual host ID of *servername*:8080)

IIS The ID corresponds exactly to the Web site name as shown in the Internet Information Services management snap-in. For example, the default Web site created when IIS is configured is named ″Default Web Site″, this is the ID used by Access Manager Plug-in for Web Servers.

iPlanet The ID corresponds exactly to the virtual host name specified when creating the virtual host in the iPlanet configuration GUI. This name is stored in the **<VS id= >** element of the **server.xml** file.

Access Manager Plug-in for Web Servers defines security policy in terms of virtual hosts. An Access Manager Plug-in for Web Servers virtual host is identified by a virtual host ID as defined above and the set of protocols (http, https or both) that it should protect. The virtual host defines the set and precedence of authentication schemes, session identification schemes and post-authorization handling that should be applied to requests to the Web server virtual host over one of the matching protocols. The virtual host also defines the mapping of URIs to Access Manager Protected Object Space names.

Access Manager Plug-in for Web Servers virtual hosts are defined in the **[pdweb-plugins]** stanza of the configuration file. They may be defined as either protected or unprotected. An unprotected virtual host will have no security policy applied to it. If a request is received that does not match one of the defined protected or unprotected virtual hosts then a warning message is generated in the Authorization Server's log file indicating the virtual host ID and the protocol of the request. This facilitates diagnosis of configuration problems.

Protected virtual hosts are defined by the **virtual-host** parameter of the **[pdweb-plugins]** stanza. Unprotected virtual hosts are defined by the **unprotected-virtual-host** parameter of the **[pdweb-plugins]** stanza. The virtual host name used typically corresponds to the virtual host ID that this virtual host matches but this is not necessarily always the case. It is the virtual host names defined in the **[pdweb-plugins]** stanza that are used to define virtual host specific security policy.

The security policy for a particular virtual host is defined by the configuration attributes specified in a stanza with the name of the virtual host. All of the attributes that may be defined in the virtual host stanza have appropriate default values so it is not necessary to have a stanza for each virtual host. It is only necessary to have such a stanza if the security policy for the virtual host differs from the default.

Two attributes of the virtual host are used to match an incoming request to the virtual host that defines the security policy that should be applied to the request. These attributes are **id** and **protocols**.

The **id** attribute is defined to be the virtual host ID that this virtual host will match. The default value for the **id** attribute is the virtual host name itself.

The protocols attribute defines the set of protocols that the virtual host will match. This value may be **http**, **https** or **both**. The default value is **both**.

The remaining attributes of the virtual host define the security policy that should be applied to requests matching this virtual host.

Virtual hosts are associated with a particular sub-branch of the protected object space. A request's URI is prefixed with this sub-branch to construct a protected object space name. This protected object space name is used for making authorization decisions. The **branch** configuration parameter defines the name of this protected object space.

```
[virtual_host_name]
branch = /PDWebPI/virtual_host_id
```

The **branch** parameter defaults to the value of the **id** parameter.

Following is an example showing the configuration parameters required for a Web server that has four virtual hosts: foo.com, bar.com-HTTP, bar.com-HTTPS and moo.com. The virtual hosts bar.com-HTTP and bar.com-HTTPS are really the same virtual host as they share the same branch; however they are distinguished by the type of access (HTTP or HTTPS). In this case authentication configuration can be set differently depending on the type of access. moo.com is not protected by the plug-in and foo.com is another virtual host on the same server.

```
[pdweb-plugins]
virtual-host = foo.com
virtual-host = bar.com-HTTPS
virtual-host = bar.com-HTTP
unprotected-virtual-host = moo.com

web-server = iplanet

[bar.com-HTTPS]
id = bar.com
protocols = https
branch = /PDWebPI/bar.com

[bar.com-HTTP]
id = bar.com
protocols = http
branch = PDWebPI/bar.com

[foo.com]
id = foo.com
protocols = http, https
branch = /PDWebPI/foo.com
```

Further configuration on a per virtual host basis is necessary to set the authentication parameters for each individual virtual host. Refer to "Setting authentication parameters for virtual hosts" on page 58 for details on configuring authentication methods for virtual hosts.

Web server specific configuration

Some actions of the plug-in are Web server specific and so particular configuration is required depending on the Web server type on which the plug-in is operating. Use the **web-server** parameter in the **[pdweb-plugins]** stanza of the `pdwebpi.conf` configuration file to define your Web server type. Valid values are **ihs**, **iplanet** or **iis**. For example:

```
[pdweb-plugins]
web-server = ihs
```

Web server specific configuration items exist in the, **[iis]**, **[iis:minimum-post-data]**, **[ihs]** and **[iplanet]** stanzas of the `pdwebpi.conf` configuration file.

The following table explains the configurable parameters for specific Web server types.

Table 5. Web server specific configuration parameters

Web Server Specific	
Parameter	**Description**
[ihs]	
query-contents	Specifies the query contents program to use for browsing the IBM HTTP Server Web space by the 'pdadmin> object list' command. This parameter can be overridden on a per branch basis by specifying a value for it in a stanza named [ihs:*branch*] e.g. [ihs:/PDWebPI/foo.bar.com]
doc-root	Specifies the documentation root which provides the Web space browse capability needed for performing 'pdadmin> object list' commands. This parameter is set by the configuration utility when setting up virtual hosts - it is specified on a per-policy branch basis in an [ihs:*branch*] stanza e.g. [ihs:/PDWebPI/foo.bar.com]
[iis]	
query contents	Specifies the query contents program for browsing the IIS Web space by pdadmin. This parameter can be overridden on a per branch basis by specifying a value for it in a stanza named [iis:*branch*] e.g. [iis:/PDWebPI/foo.com]
post-data-required	Defines a list of forms for which the submitted POST data is required for processing by the Authorization Server. For example a logon form. These parameters may NOT be overridden on a per-virtual host basis.

Table 5. Web server specific configuration parameters (continued)

Web Server Specific	
Parameter	**Description**
log-file	Defines the log file for error and trace messages from the IIS plug-in which are kept separate from the Authorization Server's log file in order to ensure consistency of the files. If specified as a relative path the location is relative to the log sub-directory of the install directory. If specified as an absolute path the absolute path is used.
[iis:minimum-post-data]	
form_uri **=** *minimum_bytes_of_post_data_required*	Defines the amount of post data for a particular form in cases when large amounts of post data are required. For example; `/token.form = 20000` Indicates that at least 20000 bytes of POST data are required by the Authorization Server when processing submissions to **/token.form**. These values cannot be specified on a per-virtual host basis.
[iplanet]	
query contents	Specifies the query contents program for browsing the iPlanet Web space by pdadmin. This parameter can be overridden on a per branch basis by specifying a value for it in a stanza named [iplanet:*branch*] e.g. [iplanet:/PDWebPI/foo.com]
doc-root	Specifies the documentation root which provides the Web space browse capability needed for performing 'pdadmin> object list' commands. This parameter is set by the configuration utility when setting up virtual hosts - it is specified on a per-policy branch basis in an [iplanet:*branch*] stanza e.g. [iplanet:/PDWebPI/foo.bar.com]

In the example below, the virtual hosts foo.com and bar.com both have a corresponding stanza in the configuration file -

```
[iplanet:/PDWebPI/foo.com]
```

and

```
[iplanet:/PDWebPI/bar.com]
```

where specific configuration parameters are defined.

```
[pdweb-plugins]
virtual-host = foo.com
virtual-host = bar.com

web-server = iplanet

[iplanet]
query-contents = /opt/pdweb/bin/wpi_iplanet_ls
```

```
[iplanet:/PDWebPI/foo.com]
doc-root = /usr/local/foo.com/doc/root

[iplanet:/PDWebPI/bar.com]
doc-root = /usr/local/bar.com/doc/root
```

Configuring plug-in auditing, logging, tracing and the cache database

Logging and auditing can provide you with information that will help in identifying any problems you maybe having with the plug-in. If you find you are having trouble and need a real-time view of error messages, then start the plug-in in the foreground using the **-foreground** option; that is,

```
pdwebpi -foreground
```

Status and error messages are logged in the file configured in the **log-file**, **logs** and **log-entries** parameters in the **[pdweb-plugins]** stanza of the `pdwebpi.conf` configuration file.

Plug-in auditing and basic cache database configuration is performed using the parameters in the **[aznapi-configuration]** stanza of the `pdwebpi.conf` configuration file.

Understanding audit records

The basic services of the authorization API allow capture of authentication (authn) and authorization (azn) audit events.

The standard 'authn' audit events do not however encapsulate sufficient information about an authentication attempt to allow the correlation of these events to a specific virtual host, where a plug-in is protecting more than a single host. For this reason, the plug-in implements its own audit event category to capture virtual host specific authentication information.

Standard 'azn' audit events do capture plug-in relevant virtual host information by virtue of the protected object name being constructed with the /PDWebPI/*virtual_host_name* prefix.

Plug-in specific authentication audit events are recorded in virtual host specific audit event pools constructed as follows:

`wpi.`*`virtual_host_name`*`.authn.`*`authentication_module_name`*

Plug-in specific authentication audit events conform to the DTD definition described in the *IBM Tivoli Access Manager Base Administrator's Guide.*

Elements of the XML style 'wpi' audit records are described in the table below:

Table 6. Authentication audit record field definitions.

XML Tag	Description
<event>	Encapsulates tag for the audit record. The element includes an attribute describing the doc type definition revision of the record.
<date>	Record of the date and time the event occurred.

Table 6. Authentication audit record field definitions. (continued)

XML Tag	Description
<outcome>	The tag element includes a **status** parameter that identifies the Access Manager or plug-in error code. The element describes the broad outcome of the event. The possible values are: • 0 = Success • 1 = Failure • 2 = Pending • 3 = Unknown
<originator>	Header tag for the originator section of the audit record. The tag element includes the **blade** parameter that identifies the Access Manager blade responsible for the event.
<component>	The tag identifies the component that captured the audit record. The component is recorded in the form: wpi.*virtual_host_name.type_of_event.module_name*
<action>	Identifies the authentication method attempted. Action codes and their corresponding authentication mechanisms are: 16961 - BA 17236 - Client side certificate 17731 - Ecsso 17999 - Failover cookie 17997 - Forms 18504 - Http Header 18768 - IP address 4806211 - IV header: PAC credential 4806229 - IV header:Username 4806220 - IV header:Distinguished name 300609 - IV header:IP address 21579 - Token
<location>	Defines the server name that actioned the event.
<accessor>	Header tag for the accessor section of the audit record. Tag element can include the name of the accessor.
<principal>	The principal tag includes the parameter **auth** which identifies the authenticating directory service. The tag defines the validated user name.
<target>	The target tag includes the parameter **resource** which can be one of the following values: • 0 = authorization • 1 = process • 2 = TCB • 3 = credential • 4 = general
<object>	Holds audit data that has little meaning for the authentication process.
<data>	Extra authentication failure information. For example, failure during an authentication attempt using HTTP header information will result in an audit log record recording the failed HTTP header in this field.

Auditing configuration

The following table displays the auditing configuration parameters and explains their function.

Table 7. Auditing configuration parameter definitions

Parameter	Description
logsize	The size (in bytes) at which the log files rollover to a new file. If set to 0 the log files will not rollover. A negative number will roll the logs over daily regardless of size.
logflush	The interval in seconds at which the logs are flushed. Maximum of 6 hours and a default of 20 seconds.
logaudit	Enables or disables auditing.
auditlog	Specifies the name of the audit file
auditcfg	Enables or disables authorization and/or authentication auditing.

For example:

```
[aznapi-configuration]
logsize = 2000000
logflush = 20
logaudit = no
auditlog = audit.log
auditcfg = azn
#auditcfg = authn
auditcfg = wpi
```

Tracing Plug-in actions

Access Manager Plug-in for Web Servers provides the ability to trace actions and store the results on file for the purposes of debugging. Primarily, tracing is an analysis and problem diagnosis tool used by application support to gain a complete view of actions causing errors. As a user, you may find some of the plug-in tracing facilities useful though the majority will be of little benefit unless you are diagnosing complex problems.

pdadmin trace commands

Listing trace components

The **list** command produces a list of all the plug-in actions that can be traced.

Syntax:

```
pdadmin> server task PDWebPI-server-name trace list [component]
```

The majority of trace tasks listed are Access Manager specific. Plug-in specific trace items are prefixed with **pdwebpi**.

Setting trace components

There are two main trace items that you may find useful for debugging:

- pdwebpi.request
- pdwebpi.plugin

pdwebpi.request set to one, traces every request that passes through the plug-in. **pdwebpi.plugin** activates trace in the plug-in server. All messages are sent to the web server's log file or in the case of IIS a log different to that used for the authorization server.

The trace **set** command has the following syntax:

```
pdadmin> server task PDWebPI-server-name trace set component
 level [file path=file|other-log-agent-config]
```

Where *component* is the name of the trace component as listed by the list command. A trace is set for this component. name. *level* is the amount of detail gathered for the trace. The range is 1 to 9. 1 being the most detailed and 9 being the least detailed. The optional **file path** parameter specifies the location for trace output. Trace output by default is sent to the standard configured plug-in log file (except when using the component **pdwebpi.plugin**). Output can be sent to the screen using the **-foreground** option. That is:

```
pdwebpi -foreground
```

Showing trace components

To show trace components use the **show** command in the following form:

```
pdadmin> server task PDWebPI-server-name trace show [component]
```

Cache database settings

You can configure the plug-in to regularly poll the master authorization database for update information. The **cache-refresh-interval** parameter can be set to ″default″, ″disable″, or a specific time interval in seconds. The ″default″ setting is **disable**.

```
[aznapi-configuration]
cache-refresh-interval = 60
```

The **db-file** parameter defines the full path to the ACL cache database. By default this is left un-set.

```
[aznapi-configuration]
db-file = /var/pdwebpi/db/pdwebpi.db
```

The **listen-flags** parameter enables or disables the reception of policy cache update notifications. A ″disable″ value disables the notification listener. This parameter is set by the `svrsslcfg` utility.

```
[aznapi-configuration]
listen-flags = disable
```

Configuring the authorization API service

The **[aznapi-entitlement-services]** stanza of the `pdwebpi.conf` configuration file assigns service ids to services. Each stanza entry defines different types of aznAPI service. For more information refer to the *IBM Tivoli Access Manager Administration C API Developer's Reference*.

Each entry is in the form

```
service_id = path_to_dll [ & params ... ]
```

Service ids are used by the aznAPI client to identify the services. You can specify parameters to pass to the service when it is initialized by the aznAPI. Parameters follow the ″&″ symbol in the entry.

The **[aznapi-admin-services]** stanza of the `pdwebpi.conf` configuration file assigns service ids to administration services. For more information refer to the *IBM Tivoli Access Manager Administration C API Developer's Reference*.

Chapter 4. IBM Tivoli Access Manager Plug-in for Web Servers authentication

This chapter discusses how IBM Tivoli Access Manager (Access Manager) Plug-in for Web Servers maintains session state, handles the authentication process and performs any post authorization processing required on authorized sessions.

Topic Index:

- "Understanding the authentication process"
- "Managing session state" on page 37
- "Managing authentication parameters" on page 43
- "Setting authentication parameters for virtual hosts" on page 58
- "Supporting Multiplexing Proxy Agents (MPA)" on page 59

Understanding the authentication process

Authentication is the method of identifying an individual process or entity that is attempting to log on to a secure domain. Successful authentication results in an Access Manager identity that represents the user. The plug-in uses this identity to acquire credentials for that user. Credentials are used by the Authorization Server to permit or deny access to protected resources.

- Access Manager Plug-in for Web Servers supports several authentication methods by default and can be customized to use other methods.
- The result of successful authentication to the plug-in is an Access Manager user registry identity.
- The plug-in uses this identity to obtain a credential for that user.
- The Authorization Server uses this credential to permit or deny access to protected objects after evaluating the ACL permissions and POP conditions governing the policy for each object.

Note: ACL = access control list policy
POP = protected object policy

During authentication, the plug-in examines a client request for the following information:

- **Virtual Host information**

  Virtual host information is included in the header of incoming requests. The plug-in uses this information to identify the virtual host addressed and match the request to Access Manager policy information.

- **Session data**

  Session data is information that identifies a specific connection between the client and the plug-in. Session data is determined from properties of the request. It is used to re-identify the client session to the plug-in and avoid the overhead of establishing a new session for each request.

- **Authentication data**

  Authentication data is information from the client that identifies the client to the plug-in. Authentication data types include client-side certificates, passwords, and token codes.

© Copyright IBM Corp. 2002

- **Post authorization data**

  Some incoming requests may be for URIs that require processing that is different to the normal case. Post authorization processing handles requests that require special authentication methods. This will often be a re-direction to a particular process designed to authenticate such requests.

The flow chart below shows the decisions made for handling requests.

For each request arriving at the Web server, the plug-in determines the virtual host the request is for and whether the virtual host is configured for protection.

Requests to virtual hosts not configured for protection are allowed to pass without further processing. For requests to virtual hosts configured for protection, the plug-in determines the identity of the user making the request. If possible, identification of the user is performed using data within the request that may be part of an existing session already having a credential assigned. In this case authorization can be performed using the existing credential. If no credential is present, the request is authorized with the **unauthenticated** credential.

If the request is authorized, the proxy determines whether modifications are required to the request or response. This processing is performed by post-authorization modules that may perform tasks such as adding headers or cookies to the request or redirecting the user to an appropriate page. If the request is not authorized using the current credential, the proxy attempts to build a new credential using authentication information (e.g. BA Headers) in the request. If it succeeds then this authentication information can be used to reattempt authorization. If there is no authentication information, the proxy attempts to build an authentication challenge response for the plug-in. If it is not possible to send the user an authentication challenge, the forbidden page is returned.

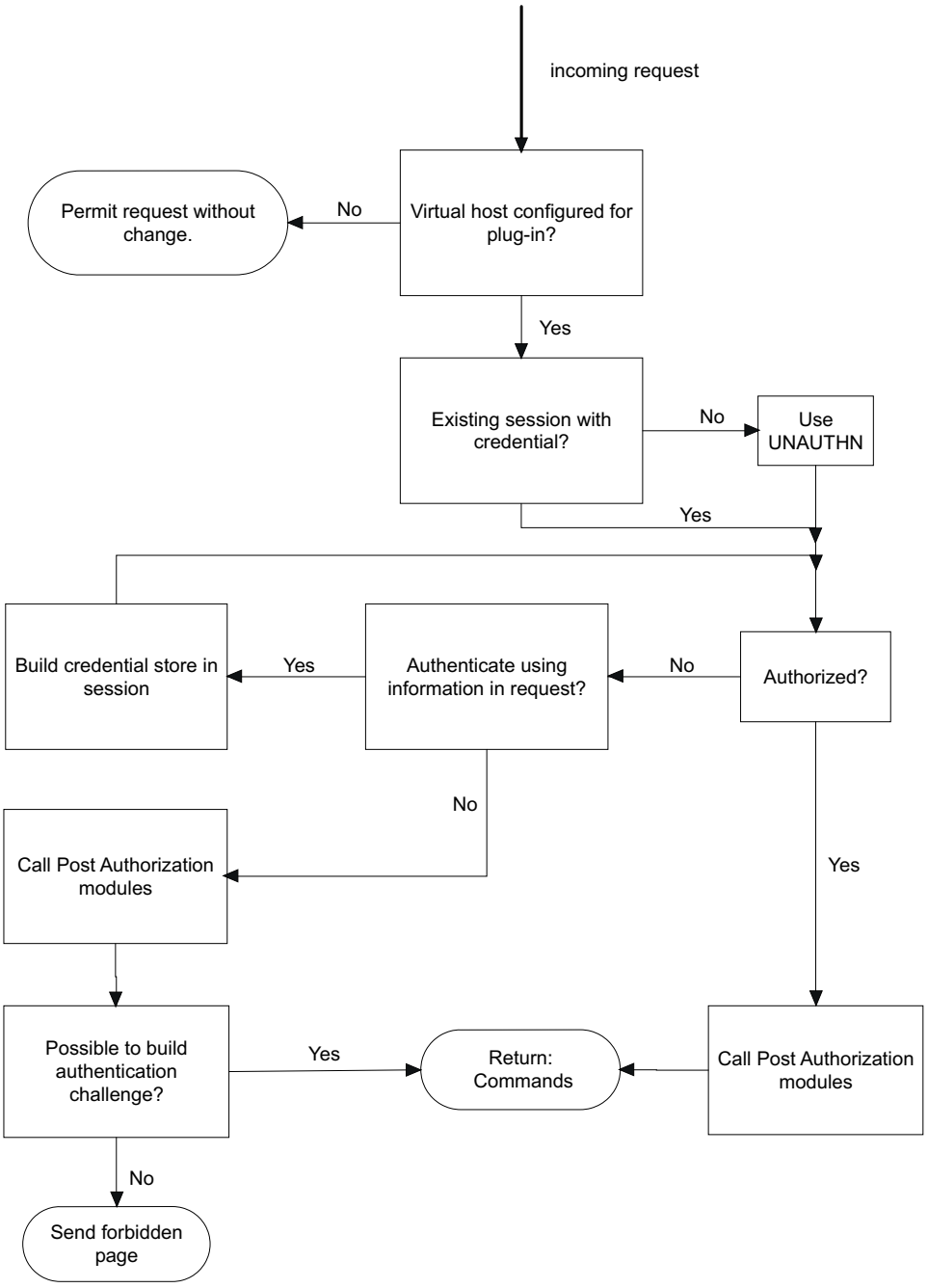

Figure 2. Web server access decisions.

Configuring authentication

All available authentication methods with their associated shared library name are defined in the **[modules]** stanza of the `pdwebpi.conf` configuration file. The **[modules]** stanza also lists the modules used for session identification and post-authorization handling. These modules are described later. The shared libraries must exist in the `pdwebpi/lib` directory. Shared library names are specified without any operating system specific prefix (e.g. lib) and any operating system specific suffix (e.g. dll). For example:

```
BA = pdwpi-ba-module
```

In the above example, the BA module library is given as `pdwpi-ba-module`. On Windows the plug-in looks for a file called `pdwpi-ba-module.dll`, on the Solaris Operating Environment it will look for a file called `libpdwpi-ba-module.so` and on AIX it will look for a file called `libpdwpi-ba-module.a`.

Note: An alternative to the default searching path for library files can be defined in the **[module-mgr]** stanza.

Each label defined in the **[modules]** stanza has a corresponding stanza of its own; for example; **[BA]**, **[cert]**and **[token]**. Specific configuration information for each authentication method is specified in these stanzas and applies to that authentication method independent of which virtual-host it is called from. If special configuration is required on a per virtual-host basis then the default configuration can be overridden by using a stanza that qualifies the module label with a virtual-host label. For example:

```
[BA]
basic-auth-realm = "Access Manager"

[BA:foo.com]
basic-auth-realm = "foo.com"
```

In the above example, users accessing virtual host foo.com using Basic Authentication will be subject to the configuration parameters specified in the stanza **[BA:foo.com]**.

The last step in configuring authentication methods is to specify the authentication methods. These are set in the **[common-modules]** stanza of the configuration file in their order of preference. For example:

```
[common-modules]
session = ssl-id
session = BA
session = session-cookie

authentication = cert
authentication = BA

post-authzn = ltpa
```

In the above example, the configuration settings ensure that:

- SSL session IDs are used to maintain session information as a first choice.
- BA headers (if available) are used to maintain session information when an SSL session ID is not available.
- Session cookies are used as a last resort to maintain session information when neither SSL session IDs or BA headers are available.
- Certificates are used as the authentication method as a first choice.
- BA is used for authentication when a certificate is not available.

- LTPA cookies are to be added to the request as part of post-authorization processing.

Configuring the order of authentication methods

The order in which configured authentication methods are displayed in the configuration file is essential to the correct operation of your plug-in software. The type of authentication methods you choose needs to be carefully considered and implemented in a way that is fail safe and achieves your security objectives.

Access Manager Plug-in for Web Servers supports a variety of authentication methods in a way that can be tailored to different customer requirements for different security needs.

As seen in previous sections of this document, the **[modules]** stanza of the `pdwebpi.conf` configuration file is where you specify the authentication methods you want to use. The **[authentication-levels]** stanza of the configuration file defines step-up authentication levels (refer to, "Authentication strength Protected Object Policy (Step-up)" on page 68) as well as the ordering of authentication methods configured in the **[modules]** stanza.

An authentication method defaults to a level of 1 when no entry for it is defined in the **[authentication-levels]** stanza. Authentication order is then determined as the highest authentication level down to the lowest authentication level for the authentication methods defined in the **[authentication-levels]** stanza. If an authentication level is shared by several modules the sub-order is then determined by the order in which the modules appear within the **[modules]** stanza.

To understand plug-in authentication it is useful to think of the plug-in asking two questions for each request it processes:

1. Can I authenticate this request using the configured method of authentication?
   If the answer to this question is no, then the plug-in asks the next question.
2. Can I generate an authentication request using the configured method of authentication?

For example; if BA is the only authentication method configured. That is:

```
[modules]
authentication = BA
```

For an incoming request authentication of the user is required if the ACL does not permit unauthorized users. The plug-in seeing BA as the only authentication method configured, asks; ″can I authenticate this request using basic authentication?″ If the request is new then the answer is no – the plug-in does not know of this user. The plug-in then asks; ″can I generate an authentication request using basic authentication?″ If basic authentication has been configured correctly, the answer is yes. The plug-in prompts the user for an ID and password.

This is a simple example of authentication using Basic Authentication. It is likely that you will want to configure more than one authentication method depending on the security requirements of your object space.

Following, is a more detailed example of the logic Access Manager Plug-in for Web Servers uses to give priority to particular authentication methods.

The authentication logic discussed in the following paragraphs assumes that unauthenticated users are not permitted to access the resource and the following configurations have been made to the pdwebpi.conf configuration file.

```
[modules]
authentication = BA
authentication = failover
authentication = forms

post-authzn = failover

[authentication-levels]
1 = BA
2 = failover
```

The above configuration specifies three authentication methods; BA, failover cookies and forms with failover cookies used for post-authorization processing. The levels set in the **[authentication-levels]** stanza determine the order in which the authentication methods are called upon to authenticate requests. Forms authentication defaults to a level of 1 as no level has been defined for it in the **[authentication-levels]** stanza.

Using the above configuration, the plug-in, when receiving a request, looks for a failover cookie in the request header. The plug-in asks the question; ″can I authenticate this request using a failover cookie?″ If the request has not been previously authenticated then the answer is no, as the plug-in will have not previously constructed a failover cookie for the request. The plug-in asks the second question, ″can I generate an authentication request using the failover cookie?″ The answer is no, as the failover cookie module has no way of generating requests for authentication.

The plug-in moves to the next configured authentication method in the **[authentication-levels]** stanza, which in the example is BA. The plug-in asks the question; ″can I authenticate this request using the BA header?″ The answer is no, as the request has not been previously authenticated. The plug-in then asks the question; ″can I generate an authentication request using BA?″ The answer is likely to be yes, and the user is prompted to enter their user ID and password. A successful authentication produces an authorized session and a failover cookie is inserted into the request header and used as the first method of authentication for subsequent requests during the same session.

Should the BA module be unable to generate a method for authenticating the user, the plug-in would default to the ordering of methods listed in the **[modules]** stanza of the configuration file. In the configuration example above, the plug-in would assign the priority of authentication methods thus:

level 1 = BA, forms
level 2 = failover cookie

If failover cookies and BA fail to provide a method for user authentication, the plug-in would authenticate using forms.

The flow chart below shows the plug-in logic used to select an authentication module.

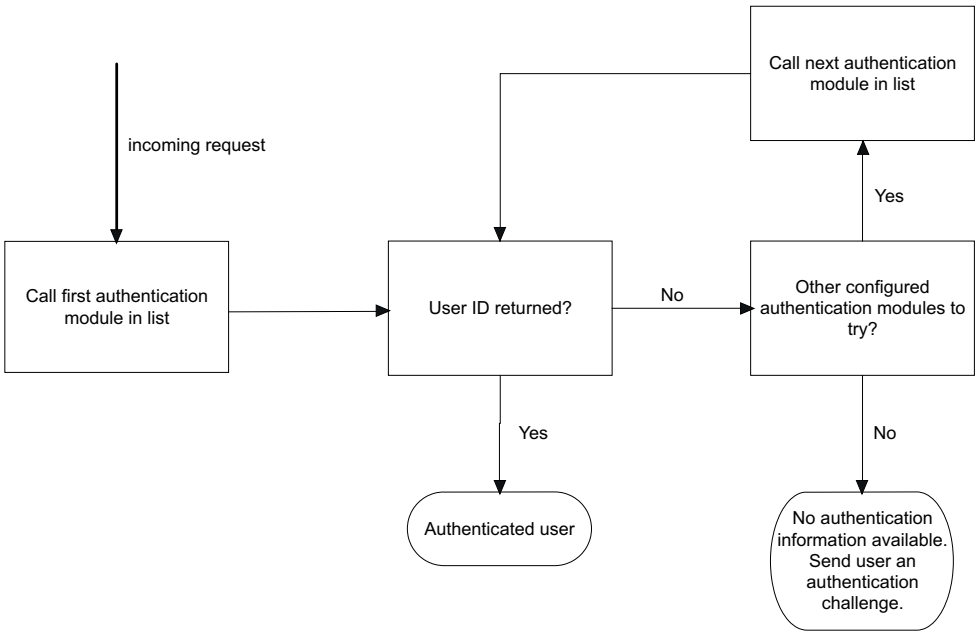

Figure 3. Plug-in process flow for determining authentication module.

The plug-in calls each authentication module in the order they are configured until one of the modules returns an Access Manager user ID. The user ID can then be used to create a credential for the user. If none of the configured authentication modules are able to provide a validated Access Manager user ID, an authentication challenge is sent to the user to prompt them to provide authentication information.

If an authentication challenge is required then the first suitable authentication module from the configured list is called to generate the commands needed (to be sent to the plug-in) to produce the challenge. Not all authentication modules can generate a challenge. For example, there is no challenge to request HTTP Headers – these are either present in the request or not. In addition, an authentication module may be unavailable because it is already being used to identify a proxy agent that is forwarding requests to the plug-in. The most common authentication mechanisms that can generate a challenge for the user are Basic Authentication (a BA challenge is sent to the user) and form-based authentication (a logon form is sent to the user). If no authentication method is available, the user cannot be authenticated and the plug-in returns a forbidden page.

The flowchart below shows the process for selecting an authentication method to send a challenge to the user.

Each configured authentication method is examined in the order in which it is configured until one is found that satisfies the required level of authentication. If a module is found that satisfies the authentication criteria it is called to build the challenge that is sent to the user. If none of the configured authentication methods are suitable then no authentication is possible. The plug-in returns a ″Forbidden″ page to the user since they do not have the permissions required to access the requested resource and there is no possibility to send them a challenge to authenticate at the required level.

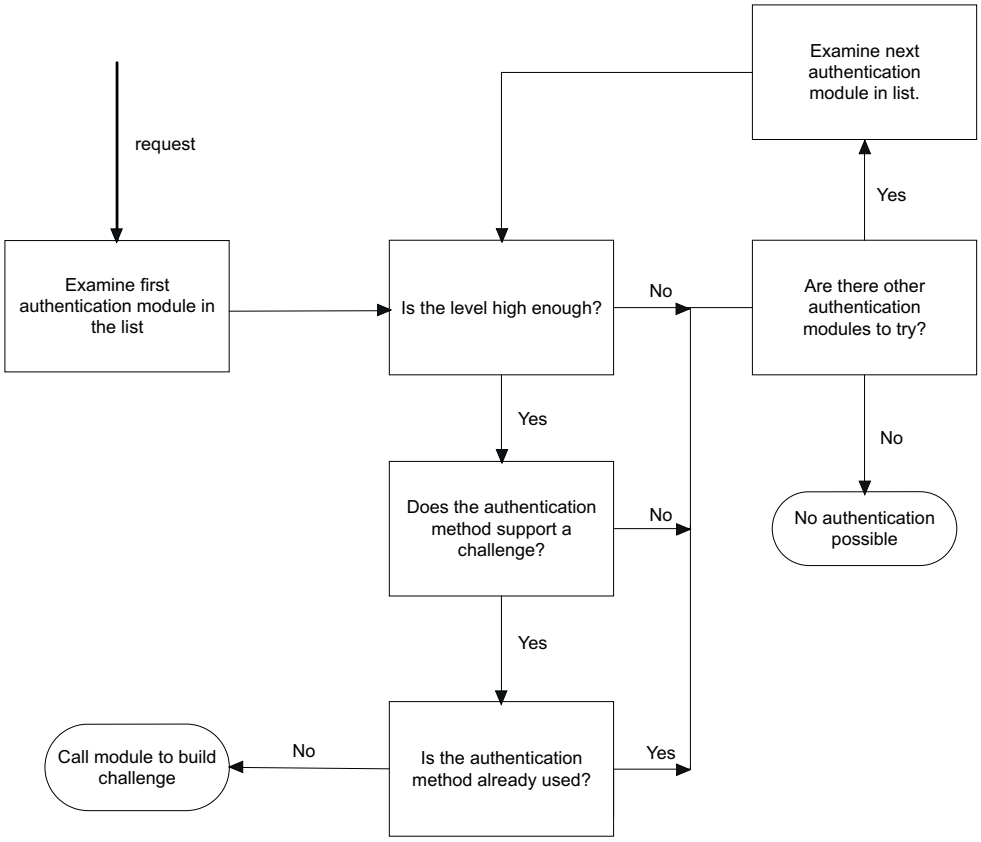

Figure 4. Authentication challenge process logic.

Configuring post-authorization processing

Configured post-authorization modules are called after a request has been authorized. Post-authorization modules determine if any other action needs to be taken before a request is passed back to the plug-in for processing by the Web server. All configured post-authorization modules are called to determine if any need to take action on the request.

Post Authorization modules are mainly of three types:

- **Modifying Request for SSO** – These post-authorization modules add information (cookies or headers) that are used by the Web application to identify the user without requiring a second authentication.
- **Modifying Response** – These post-authorization modules don't modify the request but specify that the response be altered – normally by adding headers or cookies to it. For example, the failover module adds a failover cookie to responses.
- **Special Function** – These post-authorization modules recognize the URI being requested as the trigger for some special function. This usually means that the request is handled by the plug-in. For example; eCSSO ″vouch for″ request.

Configuring authentication for virtual hosts

Configuration of authentication methods can be achieved on a per virtual host basis by specify the methods directly in each virtual host stanza. For example:

```
[pdweb-plugins]
virtual-host = foo.com
```

```
[foo.com]
....
session = ssl-id
session = BA
session = session-cookie

authentication = cert
authentication = BA

post-authzn = ltpa
```

An alternative way to specify the authentication methods for virtual hosts is to define a stanza for authentication method configuration. This allows multiple virtual-hosts to share a module configuration. The module configuration stanza is specified by the **modules** parameter in the virtual-host stanza. For example:

```
[pdweb-plugins]
virtual-host = foo.com
virtual-host = bar.com

[foo.com]
modules = foo-bar-module-stanza

[bar.com]
modules = foo-bar-module-stanza

[foo-bar-module-stanza]
authentication = ba
session = ba
post-authzn = ltpa
```

When separate stanzas for authentication methods configuration on a per-virtual host basis are not defined in the configuration file, all virtual hosts use the parameters configured in the **[common-modules]** stanza.

Managing session state

Session state information is used by the plug-in to identify the source of incoming requests. The plug-in uses the identity of the request source to maintain the session state between client and server when the client performs numerous requests within the one session. Without an established session state between client and server, the communication between the client and the server must be renegotiated for each subsequent request. Session state information improves performance by eliminating the need for repeated authentication. The client can log on once and make numerous requests without performing a separate log on for each request.

Access Manager Plug-in for Web Servers handles both HTTP and HTTPS communication. The plug-in is designed to use any of the following information types to maintain session state with a client:

1. SSL Session ID
2. Basic Authentication
3. Server-specific session cookie
4. HTTP header data
5. IP address

The plug-in calls each configured session module in turn. The plug-in continues to search the configured session module types until one returns a credential. The plug-in then determines if the application references a Multiplexing Proxy Agent. If it is a Proxy Agent then another session must exist for the real end user. To find

this other session the plug-in continues to call the rest of the configured session modules. A user credential is returned when an existing session is found for which user authentication has already taken place. This credential is used to authorize the request. If none of the configured session modules return a user credential, the session is either new or is a session for which no credential has been established.

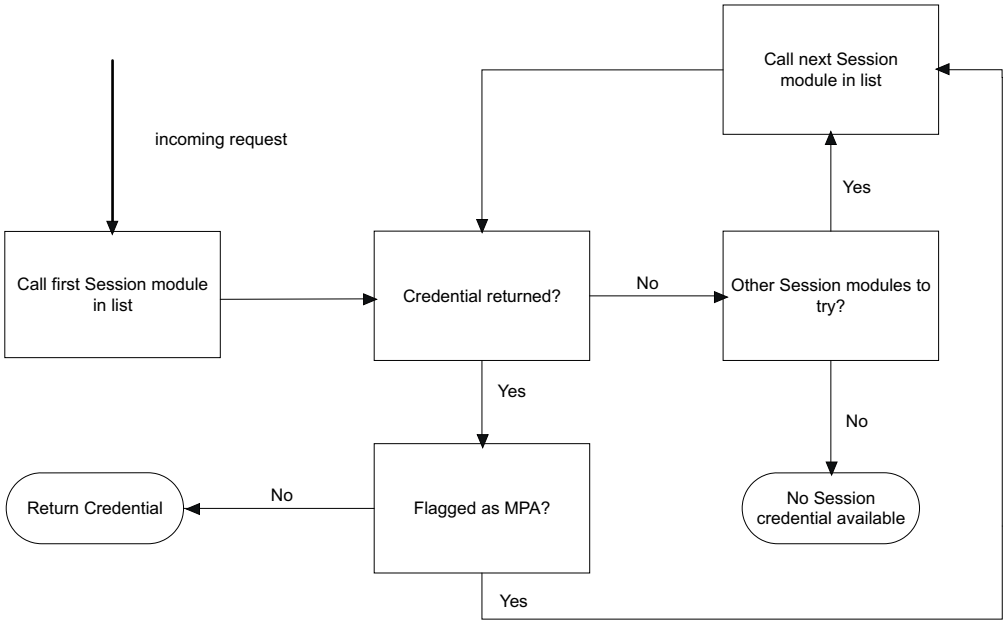

Figure 5. Plug-in process flow for determining session module.

Configuring the plug-in session/credentials cache

The plug-in session cache allows a server to store the session ID information from multiple clients. The session cache accommodates both HTTPS and HTTP session state information.

The plug–in cache stores session ID information plus the credential information obtained for each client. Credential information is cached to eliminate repetitive queries to the user registry database during authorization checks. The plug-in cache also maintains session state information for SSL connections between the plug-in and the LDAP user registry.

There are several configuration parameters available for the plug-in cache that allow you to tune the performance of the cache.

Note: The values configured in the **[sessions]** stanza of the `pdwebpi.conf` configuration file may be overridden in the **[***module_name***]** stanza, and some may be further overridden in the **[***module_name:virtual_host_name***]** stanza on a per-virtual host basis.

Setting the maximum concurrent entries value

The **max-entries** parameter, located in the **[sessions]** stanza of the `pdwebpi.conf` configuration file, sets the maximum number of concurrent entries in the plug-in's session/credentials cache.

This value corresponds to the number of concurrent logon sessions. When the cache size reaches this value, entries are removed from the cache according to a least recently used algorithm to allow new incoming logons.

The default number of concurrent logon sessions is 4096:

```
[sessions]
max-entries = 4096
```

Setting the cache entry timeout value

The **timeout** parameter, located in the **[sessions]** stanza of the `pdwebpi.conf` configuration file, sets the maximum lifetime timeout for an entry in the plug-in's session / credentials cache.

The plug-in caches credential information internally. The session cache timeout parameter dictates the length of time authorization credential information remains in memory.

The parameter is not an inactivity timeout. The value maps to a ″credential lifetime″ rather than a ″credential timeout″. Its purpose is to enhance security by forcing the user to re-authenticate when the specified timeout limit is reached.

The default logon session timeout (in seconds) is 7200:

```
[sessions]
timeout = 7200
```

You can configure the session cache lifetime to be reset whenever reauthentication occurs. Each time reauthentication occurs, the session cache **timeout** value is reset. To configure session cache lifetime reset use the **reauth-lifetime-reset** parameter in the **[sessions]** stanza of the `pdwebpi.conf` configuration file:

```
[sessions]
reauth-lifetime-reset = yes
```

The default value is ″no″.

It is possible for the session cache lifetime value to expire while the user is performing a reauthentication. The session cache lifetime can expire after the reauthentication logon form is sent to the user and before the completed logon form is returned. When the session cache lifetime value expires, the session cache entry is deleted. When the logon form is returned to the plug-in, there is no longer a session for that user. In addition, all cached user request data is lost. You can configure a time extension, or ″grace period″, for the session cache lifetime should the session cache lifetime expire during reauthentication.

The **reauth-grace-period** parameter in the **[sessions]** stanza of the `pdwebpi.conf` configuration file provides this time extension, in seconds. For example:

```
[reauthentication]
reauth-grace-period = 20
```

The default value, ″0″, provides no extension to the session cache timeout value. The **reauth-grace-period** parameter applies to users with existing session cache entries and who are required to reauthenticate. For example:

- Users performing reauthentication resulting from POP security policy
- Users performing reauthentication resulting from session cache inactivity
- Users performing step-up authentication

The **reauth-grace-period** option is intended to be used in conjunction with the **reauth-lifetime-reset** = yes option.

Setting the cache entry inactivity timeout value

The **inactive-timeout** parameter, located in the **[sessions]** stanza of the pdwebpi.conf configuration file, sets the timeout value for logon session inactivity.

The default logon session inactivity timeout (in seconds) is 3600:

```
[sessions]
inactive-timeout = 3600
```

To disable this timeout feature, set the parameter value to ″0″.

Maintaining session state with the SSL session ID

Access Manager Plug-in for Web Servers can track sessions using the SSL session ID of incoming HTTPS requests. This facility is not available on IIS, as IIS does not make SSL session IDs available to the plug-in.

Note: SSL session IDs are not used for authenticating requests.

The **[common-modules]** stanza in the pdwebpi.conf configuration file defines the use of all session, authentication and post-authorization methods using the format *module_type* = *module-name*. To maintain session state using SSL session ID's, assign the word **ssl-id** to the **session** parameter as in the following:

```
[common-modules]
session = ssl-id
```

Ensure the shared library for ssl-id is configured in the **[modules]** stanza of the pdwebpi.conf configuration file. That is:

```
[modules]
ssl-id = pdwpi-sslsessid-module
```

Maintaining session state using Basic Authentication

Basic Authentication (BA) is a method for authenticating users and maintaining session state via the input of a user-name and password. BA is defined by the HTTP protocol and can be implemented over HTTP and HTTPS.

Basic Authentication maintains session state by caching a record of the content of the Basic Authentication header.

To configure the plug-in to maintain session state using Basic Authentication, use the **[common-modules]** stanza in the pdwebpi.conf configuration file. Enter the key-word **session** with the value as **BA**, as in the following:

```
[common-modules]
session = BA
```

If BA is used to maintain session state, it needs to be also used for user authentication. The **[common modules]** stanza of the configuration file should also set BA for authentication.

```
[common-modules]
session = BA
authentication = BA
```

The **[BA]** stanza in the pdwebpi.conf file defines the Basic Authentication realm. The realm is the text that is displayed in the dialog box that is displayed when the browser prompts the user for logon data.

```
[BA]
basic-auth-realm = realm_name
```

Maintaining session state with Session Cookies

Using session cookies to hold session information is a method for maintaining session state, though it should only be used when no other mechanisms are available. The server packages the state information for a particular client in a cookie and sends it to the client's browser. For each new request, the browser re-identifies itself by sending the cookie (with the session information) back to the server.

Session cookies offer a possible solution for situations when the client uses a browser that renegotiates its SSL session after very short periods of time. For example, some versions of the Microsoft Internet Explorer browser renegotiate SSL sessions every two or three minutes.

A session cookie only provides re-authentication of a client to the single, unique server that the client had previously authenticated to within a short time period (around ten minutes). The mechanism is based on a ″server cookie″ that cannot be passed to any machine other than the one which generated the cookie.

In addition, the session cookie contains only a random number identifier that is used to index the cookie in the server's session cache. There is no other information exposed in the session cookie. The session cookie cannot compromise security policy.

Access Manager Plug-in for Web Servers uses a secure server-specific session cookie. The following conditions apply to this cookie mechanism:

- Cookie contains session information only; it does not contain identity information.
- Cookie resides only in the browser memory (it is not written to the browser cookie jar on the disk).
- Cookie has a limited lifetime (configurable).
- Cookie has path and domain parameters that prohibit its use by other servers.

To configure the plug-in to use session cookies to maintain session state, use the **[common-modules]** stanza in the `pdwebpi.conf` configuration file. Enter the key-word **session** with the value as **session-cookie**, as in the following:

```
[common-modules]
session = session-cookie
```

The **resend-pdwebpi-cookies** parameter, located in the **[sessions]** stanza of the `pdwebpi.conf` configuration file, enables or disables the sending of the session cookie to the browser with every response. This action helps to ensure that the session cookie remains in the browser memory. The **resend-pdwebpi-cookies** parameter has a default setting of ″no″:

```
[sessions]
resend-pdwebpi-cookies = no
```

Change the default setting to ″yes″ to send plug-in session cookies with every response.

Maintaining session state using HTTP headers

Access Manager Plug-in for Web Servers can be configured to use HTTP header information to identify sessions and maintain session state.

To specify multiple HTTP headers, multiple instances of the HTTP Header module must be configured. For example:

```
[modules]
entrust-client-header = pdwpi-httphdr-module
some-other-header     = pdwpi-httphdr-module

[entrust-client-header]
header = entrust-client

[some-other-header]
header = some-other
```

The plug-in can use HTTP headers for tracking sessions as well as authenticating users. If the plug-in is configured to use HTTP headers to track sessions then it must also be configured to use HTTP headers to authenticate users. However, having the plug-in configured to authenticate incoming requests using HTTP headers does not require the plug-in to be configured to track sessions. Refer to, “Configuring HTTP header authentication” on page 55 for details on configuring the plug-in to use HTTP headers for client authentication.

When using HTTP headers to maintain session state, the **[common-modules]** stanza of the `pdwebpi.conf` configuration file will have the following entries:

```
[common-modules]
authentication = http-hdr
session = http-hdr
```

Maintaining session state using IP addresses

Access Manager Plug-in for Web Servers can use IP addresses to identify and track sessions.

To configure the plug-in to use IP addresses to track sessions, use the **[common-modules]** stanza in the `pdwebpi.conf`. Enter the key-word **session** with the value as **ip-addr**. That is:

```
[common-modules]
session = ip-addr
```

Ensure the shared library for IP address authentication is configured in the **[modules]** stanza of the `pdwebpi.conf` configuration file. That is:

```
[modules]
ip-addr = pdwpi-ipaddr-module
```

If IP address are used to maintain session state they must also be used to authenticate incoming requests. See “Configuring IP address authentication” on page 56 for details on configuring Access Manager Plug-in for Web Servers to use the IP address as the method for client authentication. The usage of IP addresses for authenticating clients, however, does not require them to be used as the method for identifying sessions.

Managing authentication parameters

Authentication configuration overview

The mechanisms for all authentication methods supported by Access Manager Plug-in for Web Servers are configured in the **[authentication-mechanisms]** stanza of the `pdwebpi.conf` configuration file. Supported authentication method parameters include:

- Local (built-in) authenticators

  Parameters for local authenticators specify the appropriate built-in shared library (UNIX) or DLL (Windows) files.
- Custom external authenticators

  The plug-in provides template server code that you can use to build and specify a custom external Cross Domain Authentication Service (CDAS) server.

  An external CDAS authenticator specifies the appropriate custom shared library.

Local authentication parameters

The following parameters specify local built-in authenticators:

Table 8. Local Built-in Authenticators.

Parameter	Description
Forms and Basic Authentication	
passwd-ldap	Client access with LDAP username and password.
Client-side Certificate Authentication	
cert-ssl	Client access with client-side certificate over SSL.
HTTP Header and/or IP Address Authentication and/or IV Header with iv-remote-address activated.	
http-request	Client access via special HTTP header and/or IP address and/or IV Header with iv-remote-address activated.

Use the **[authentication-mechanisms]** stanza to configure the authentication method and the implementation in the following format:

authentication_method_parameter = shared_library

External custom CDAS authentication parameters

The following parameters are available to specify custom shared libraries for external CDAS servers:

Table 9. External CDAS Server Parameters.

Parameter	Description
passwd-cdas	Client access with username and password for a third-party registry.
token-cdas	Client access with LDAP username and token passcode.
cert-cdas	Client access with client-side certificate over SSL.

In addition to the authentication libraries there are two other standard Access Manager libraries that can be used in the plug-in:

- **passwd-strength**

  This library checks new passwords entered on the password change form.
- **cred-ext-attrs**

This library allows custom attributes (name/value pairs) to be specified for inclusion in the credential.

Refer to the *IBM Tivoli Access Manager WebSEAL Developer's Reference* for details on building and configuring a custom shared library that implements a CDAS server.

Default configuration for plug-ins

By default the plug-in is set to authenticate clients using Basic Authentication (BA) usernames and passwords (LDAP registry).

The plug-in is normally enabled for both TCP and SSL access. Therefore, a typical configuration of the **[authentication-mechanisms]** stanza includes support for username and password (LDAP registry) and support for client-side certificates over SSL.

The following example represents the typical configuration of the **[authentication-mechanisms]** stanza for the Solaris Operating Environment:

```
[authentication-mechanisms]
passwd-ldap = libldapauthn.so
cert-ssl = pdwpi-sslauthn.so
```

To configure other authentication methods, add the appropriate parameter with its shared library (or CDAS module).

Configuring multiple authentication methods

You modify the **[authentication-mechanisms]** stanza of the `pdwebpi.conf` configuration file to specify the shared library to be used for any supported authentication method. The following conditions apply when you configure multiple authentication methods:

1. All authentication methods can function independently from each other. It is possible to configure a shared library for each supported method.
2. The **cert-cdas** method overrides the **cert-ssl** method when both are configured. You must enable one of these to support client-side certificates.
3. Only one password type authenticator is actually used when more than one is configured. The plug-in uses the following order of priority to resolve multiple configured password authenticators:
   a. **passwd-cdas**
   b. **passwd-ldap**
4. It is possible to configure the same custom library for two different authentication methods. For example, you could write a custom shared library to process both username/password and HTTP header authentication. For this example, you would configure both the **passwd-cdas** and **http-request** parameters with the same shared library. It is the responsibility of the developer to maintain session state and avoid conflicts between the two methods.

Prompting for log on

The plug-in prompts a client for a log on under the following conditions:

1. An unauthenticated client fails an authorization check
2. A Forms or Basic Authentication client fails an authorization check

The following client types are presented a ″403 failure″ error:

1. When an authorization check fails:
   a. Client-side certificate
   b. Failover cookie

c. CDSSO
d. IP address
e. HTTP header

2. When a client authenticates with a method that is disabled by the plug-in

Logout, change of password and help commands

Access Manager provides the following commands for supporting clients who authenticate over HTTP or HTTPS.

pkmslogout: Clients can use the **pkmslogout** command to log out from the current session when they use an authentication method that does not supply authentication data with each request. For example, **pkmslogout** does not work for clients using Basic Authentication or IP address authentication. In this case, you must close the browser to log out.

The **pkmslogout** command is appropriate for authentication via client-side certificate, token passcode, Forms authentication, and certain implementations of HTTP header authentication.

Run the command as follows:

```
https://www.tivoli.com/pkmslogout
```

The browser displays a logout form defined in the `pdwebpi.conf` configuration file:

```
[acctmgmt]
logout-uri = /pkmslogout
logout-success = logout_success.html
```

You can modify the `logout_success.html` file to suit your requirements.

The **pkmslogout** utility also supports multiple logout response pages when the network architecture requires different exit screens for users logging out of distinctly different virtual hosts.

pkmspasswd: You can use this command to change your logon password when using Basic Authentication (BA) or Forms authentication. This command is appropriate over HTTP or HTTPS.

For example:

```
https://www.tivoli.com/pkmspasswd
```

The browser displays a change of password form defined in the `pdwebpi.conf` configuration file:

```
[acctmgmt]
password-change-form-uri = /pkmspasswd.form
password-change-uri = /pkmspasswd
password-change-success = password_change_success.html
password-change-failure = password_change_failure.html
```

You can modify the `password_change_success.html` and `password_change_failure.html` files to suit your requirements.

pkmshelp: You can use this command to access help pages. This command is appropriate over HTTP or HTTPS.

The name and location of help pages are defined in the `pdwebpi.conf` configuration file:

```
[acctmgmt]
help-uri = /pkmshelp
help-page = help.html
```

You can modify the help.html file to suit your requirements.

Configuring Basic Authentication

Basic Authentication (BA) is a standard method for providing a username and password to the authentication mechanism. BA is defined by the HTTP protocol and is implemented over HTTP and over HTTPS.

Enabling Basic Authentication

By default, the plug-in is configured for BA username and password. The **[common-modules]** stanza in the pdwebpi.conf configuration file defines the use of BA for authenticating requests. That is:

```
[common-modules]
authentication = BA
```

The **[modules]** stanza in the pdwebpi.conf configuration file defines all available authentication mechanisms and their associated shared library name. Ensure that the entry for basic authentication exists; that is:

```
[modules]
BA = pdwpi-ba-module
```

By default the BA authentication mechanism is given a level of one in the **[authentication levels]** stanza of the configuration file. This setting relates to the priority of authentication mechanisms for incoming requests.

Setting the realm name

The realm is displayed in the dialog presented by a browser to the user when challenging them for a username and password. The realm name is assigned to the **basic-auth-realm** parameter in the **[BA]** stanza of the pdwebpi.conf configuration file.

```
[BA]
basic-auth-realm = realm_name
```

Configuring the Basic Authentication mechanism

The **passwd-ldap** parameter specifies the shared library used to handle username and password authentication.

- On UNIX, the file that provides the built-in mapping function is a shared library called libldapauthn.
- On Windows, the file that provides the built-in mapping function is a DLL called ldapauthn.

Table 10. BA shared library authentication mechanisms

	Shared Library		
Authn Mechanisms	**Solaris Operating Environment**	**AIX**	**Windows**
passwd-ldap	libldapauthn.so	libldapauthn.a	ldapauthn.dll

You can configure the username and password authentication mechanism by entering the **passwd-ldap** parameter with the platform-specific name of the shared library file in the **[authentication-mechanisms]** stanza of the pdwebpi.conf configuration file – as in the following:

Solaris Operating Environment:

```
[authentication-mechanisms]
passwd-ldap = libldapauthn.so
```

Windows:

```
[authentication-mechanisms]
passwd-ldap = ldapauthn.dll
```

Configuring forms authentication

Access Manager provides Forms authentication as an alternative to the standard Basic Authentication mechanism. This method produces a custom HTML logon form from Access Manager instead of the standard logon prompt resulting from a Basic Authentication challenge.

When you use Forms-based logon, the browser does not cache the username and password information as it does with Basic Authentication.

Enabling forms authentication

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable authentication via forms, assign the word 'forms' to the authentication parameter; that is:

```
[common-modules]
authentication = forms
```

When using forms for authentication, the plug-in must also be configured to use forms for post-authorization processing. This allows the plug-in to redirect the authenticated user back to the original request URL. In the **[common-modules]** stanza of the `pdwebpi.conf` configuration file, add the parameter **post-authzn** as in the following:

```
[common-modules]
authentication = forms
post-authzn = forms
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library name. Ensure that the entry for forms authentication exists; that is:

```
[modules]
forms = pdwpi-forms-module
```

Configuring the forms authentication mechanism

The **passwd-ldap** parameter specifies the shared library used to handle username and password authentication.

- On UNIX, the file that provides the built-in mapping function is a shared library called `libldapauthn`.
- On Windows, the file that provides the built-in mapping function is a DLL called `ldapauthn`.

Table 11. Forms shared library authentication mechanism

Authn Mechanisms	Shared Library		
	Solaris Operating Environment	**AIX**	**Windows**
passwd-ldap	`libldapauthn.so`	`libldapauthn.a`	`ldapauthn.dll`

You can configure the username and password authentication mechanism by entering the **passwd-ldap** parameter with the platform-specific name of the shared library file in the **[authentication-mechanisms]** stanza of the pdwebpi.conf configuration file, as in the following:

Solaris Operating Environment:

```
[authentication-mechanisms]
passwd-ldap = libldapauthn.so
```

Windows:

```
[authentication-mechanisms]
passwd-ldap = ldapauthn.dll
```

Customizing HTML response forms

Forms authentication requires you to use a custom logon form. By default, the sample login.html form is located in the following directory:
install_directory/nls/html/*lang*

Where *lang* is taken from the NLS configuration. In a US English installation this will be set to **C**.

The **login-form** parameter in the **[forms]** stanza of the configuration file defines the file name of the form presented to the user during log on. The path of the file should be relative to the translated pdwebpi HTML directory; e.g. *pdwebpi*/nls/html/*lang*.

```
[forms]
login-form = login.html
```

Configuring certificate authentication

Access Manager Plug-in for Web Servers supports secure communication with clients using client-side digital certificates over SSL. In this authentication method, certificate information (such as the Distinguished Name or DN) is mapped to an Access Manager identity.

Mutual authentication via certificates

Authentication via digital certificates takes place in two stages:

- The Web server where the plug-in is resident identifies itself to SSL clients with its server-side certificate.
- The Web server uses its database of Certificate Authority (CA) root certificates to validate clients accessing with client-side certificates.

1. An SSL client requests a connection with a Web server via the plug-in.
2. In response, the Web server sends its public key via a signed server-side certificate. This certificate has been previously signed by a trusted third-party certificate authority (CA).
3. The client checks whether the certificate's issuer is one that it can trust and accept. The client's browser usually contains a list of root certificates from trusted CAs. If the signature on the Web server's certificate matches one of these root certificates, then the server can be trusted.
4. If there is no match for the signature, the browser informs its user that this certificate was issued by an unknown certificate authority. It is then the user's responsibility to accept or reject the certificate.
5. If the signature matches an entry in the browser's root certificate database, session keys are securely negotiated between the client and the Web server.

The end result of this process is a secure channel over which the client can authenticate (for example, via username and password). After successful authentication, the client and server can continue to communicate securely over this channel.

6. Now the client sends its public key certificate via the plug-in to the Web server.
7. The Web server attempts to match the signature on the client certificate to a known CA using the Web server's certificate store.
8. If there is no match for the signature, an SSL error code is generated and sent it to the client.
9. If there is a match for the signature, then the client can be trusted. Authentication of the client takes place, resulting in a Access Manager identity.
10. Session keys are securely negotiated between the client and the Web server. The end result of this process is a secure and trusted communication channel between the mutually authenticated client and server.

Enabling certificate authentication

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable authentication via certificates, assign the word 'cert' to the **authentication** parameter; that is:

```
[common-modules]
authentication = cert
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and the associated shared library name. Ensure that the entry for certificate authentication exists; that is:

```
[modules]
cert = pdwpi-certificate-module
```

Configuring the certificate authentication mechanism

The **cert-ssl** parameter specifies the shared library for mapping certificate authentication information.

On UNIX, the file that provides the built-in mapping function is a shared library called `libpdwpi-sslauthn`. On Windows, the file that provides the built-in mapping function is a DLL called `sslauthn`.

Table 12. Certificate shared library authentication mechanisms

Authn Mechanisms	Shared Library		
	Solaris Operating Environment	**AIX**	**Windows**
cert-ssl	`libpdwpi-sslauthn.so`	`libpdwpi-sslauthn.a`	`pdwpi-sslauthn.dll`

You can configure the certificate authentication mechanism by entering the **cert-ssl** parameter with the platform-specific name of the shared library file in the **[authentication-mechanisms]** stanza of the `pdwebpi.conf` configuration file.

Solaris Operating Environment:

```
[authentication-mechanisms]
cert-ssl= libpdwpi-sslauthn.so
```

Windows:

```
[authentication-mechanisms]
cert-ssl = pdwpi-sslauthn.dll
```

The default mapping provided by the shared library file directly maps a certificate DN to an LDAP DN.

Configuring token authentication

Access Manager Plug-in for Web Servers supports authentication via a token passcode supplied by the client. This authentication uses a two factor log on based on RSA SecureID® fobs.

Enabling token authentication

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable authentication via tokens, assign the word 'token' to the **authentication** parameter.

When authentication using tokens is enabled then tokens must also be configured for post-authorization processing. In the **[modules]** stanza of the configuration file, construct a **post-authzn** parameter and assign it the value 'token'. The **[common-modules]** stanza should include the following two entries:

```
[common-modules]
authentication = token
post-authzn = token
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and the associated shared library name. Ensure that the entry for token authentication exists; that is:

```
[modules]
token = pdwpi-token-module
```

Configuring the token authentication mechanism

The **token-cdas** parameter specifies the shared library for mapping token passcode authentication information.

- On UNIX, the file that provides the built-in mapping function is a shared library called `libtokenauthn`.
- On Windows, the file that provides the built-in mapping function is a DLL called `tokenauthn`.

Table 13. Token shared library authentication mechanisms

Authn Mechanisms	Shared Library		
	Solaris Operating Environment	**AIX**	**Windows**
token-cdas	`libtokenauthn.so`	`libtokenauthn.a`	`tokenauthn.dll`

By default, this built-in shared library is hard-coded to map SecurID token passcode data. You can customize this file to authenticate other types of special token data and, optionally, map this data to an Access Manager identity. Refer to the *IBM Tivoli Access Manager WebSEAL Developer Reference* for API resources.

You can configure the token authentication mechanism by entering the **token-cdas** parameter with the platform-specific name of the shared library file in the **[authentication-mechanisms]** stanza of the `pdwebpi.conf` configuration file.

For example:

Solaris Operating Environment:

```
[authentication-mechanisms]
token-cdas = libtokenauthn.so
```

Windows:

```
[authentication-mechanisms]
token-cdas = tokenauthn.dll
```

Customizing token response pages

The **token-login-form** parameter in the **[token-card]** stanza of the configuration file defines the file name of the form presented to the user client during a token logon. The path of the file should be relative to the translated pdwebpi HTML directory; e.g. *pdwebpi*/nls/html/*lang*. Where *lang* is taken from the NLS configuration. In a US English installation this will be set to **C**.

The **next-token-form** parameter in the **[token-card]** stanza defines the form displayed to the user client to request the next token. The client is requested to enter another token when the server can not successfully authenticate the user from the first. Inability to authenticate the user may be caused by a number of reasons, most commonly though it is due to the client and server clocks not being synchronized. When authentication can not succeed using the first token, the page specified in the **next-token-form** parameter is displayed to prompt for the next token.

The **token-card** stanza has the following format:

```
[token-card]
token-login-form = tokenlogin.html
next-token-form = nexttoken.html
```

Configuring failover cookie authentication

The failover cookie feature is typically used for clients connecting to a replicated front-end web server through a load-balancing mechanism. A failover cookie prevents forced re-authentication when the original session between server and client becomes unavailable.

With failover cookies configured for post-authorization processing, the plug-in encrypts credential data in either a server specific or domain wide cookie. The cookie is placed on the browser when the client first connects. If the initial web server session is lost, the cookie is presented to the next server the client is redirected to. The cookie is used for automatic re-authentication so the client is spared the task of re-authenticating manually. The plug-ins on replicated servers share a common key that decrypts the credential information held in the cookie and establishes a new session.

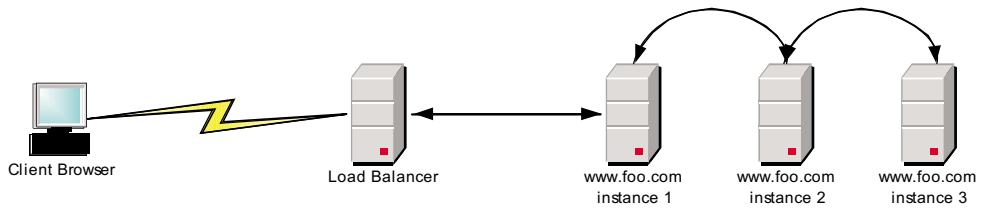

Figure 6. Typical server architecture for failover cookies.

The diagram above shows a typical architecture that would benefit from the use of failover cookies. Three identical instances of the same Web server are located

behind a load balancing server that directs requests to one of the three servers depending on load and availability. For example, assume that each instance of www.foo.com is configured to authenticate client accesses using failover cookies and also configured to use failover cookies for post-authorization processing. A client accesses www.foo.com and is directed to instance 1 of the server and authenticates successfully. The client's credential is encrypted and stored in a domain wide cookie that is stored at the client browser. If during the session the client needs to accesses either instance 2 or instance 3 of www.foo.com (e.g. if instance 1 fails or demand becomes too great) then the failover cookie stored in the client's browser is used for automatic re-authentication without the need for user intervention.

Enabling authentication using failover cookies

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. Failover cookies can be configured to perform authentication and post-authorization tasks.

Plug-ins configured for post-authorization processing using failover cookies, encrypt and store a credential as a failover cookie in the transaction response.

Plug-ins, configured to use failover cookies for performing authentication, re-authenticate clients using the encrypted credential from a failover cookie found in the transaction request.

To enable authentication and post-authorization using failover cookies, assign the reference 'failover' to the **authentication** and **post-authzn** parameters; that is:

```
[common-modules]
authentication = failover
post-authzn = failover
```

Note: Failover cookie authentication must be configured as the initial authentication method when other authentication mechanisms are configured along with failover cookies.

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library name. Ensure that the entry for failover cookie authentication exists; that is:

```
[modules]
failover = pdwpi-failovercookie-module
```

Configuring failover cookie parameters

Failover cookie authentication parameters are configured in the **[failover]** stanza of the `pdwebpi.conf` configuration file.

The **failover-cookies-keyfile** parameter specifies the file that is used to encrypt and decrypt credential data in the failover cookie. For example:

```
[failover]
failover-cookies-keyfile = failover.key
```

The key file must be created using the program **pdwpi-cdsso-key-gen** located in the *install_path*`/bin` directory. Usage:

`./pdwpi-cdsso-key-gen` *key_file_name_to_create*

The **failover-cookies-lifetime** parameter defines the valid lifetime of the failover-cookie in minutes. This is the time between cookie creation and the cookie being disabled. The default value is 30 minutes.

```
[failover]
failover-cookies-lifetime = 30
```

The **enable-failover-cookie-for-domain** parameter enables or disables the cookie validity within the entire domain.

For example:

```
[failover]
enable-failover-cookie-for-domain = false
```

Configuring IV header authentication

Access Manager supports authentication via internally generated header information supplied by a compatible client or a proxy agent. For historic reasons these are called IV (IntraVerse) headers. When the plug-in enhanced Web server receives requests from a trusted application such as WebSEAL or a multi–plexing proxy agent, IV headers may be inserted into the requests relayed to the plug-in proxy server. IV headers contain information that identify the originating client rather than the relaying server. The information in the headers is used to construct an originating client credential for authorization purposes. Similarly, if the plug-in enhanced web server relays requests to another Access Manager server that recognizes IV headers, the plug-in proxy can insert IV headers to identify the originating client.

The plug-in can be configured to use IV headers for post-authorization processing or for authenticating requests. Configured for post-authorization processing , the plug-in, after a successful authentication, modifies a transaction request by inserting the client's true identity as IV headers. These headers may then be forwarded on to another server by the originating web server.

If the plug-in is configured to use IV Headers to perform client authentication, the plug-in creates a client credential using the identity extracted from an IV header found in a transaction request. Since it is easy for clients to fake IV headers, such a credential is only created if the proxy server nominates trust of the headers by setting the 'use secondary authenticator' flag in the authenticate request.

For authentication, IV headers can be configured to accept one, some, or all of iv-user, iv-user-l, iv-creds or iv-remote-address headers in the request as proof of authentication when received via a proxy. The iv-remote-address header is used to record the real remote address of the user.

Configured for post-authorization processing, IV headers are inserted with one, some or all of the iv-user, iv-user-l, iv-creds, iv-groups, and/or iv-remote-address, HTTP headers into the request.

Table 14. IV header field descriptions

IV Header Field	Description
iv-user	The short name of the Access Manger user. Defaults to unauthenticated if the client is unauthenticated (unknown).
iv-user-l	The full domain name of the user (long form). E.g. LDAP distinguished name.
iv-groups	A list of the groups to which the user belongs.
iv-creds	Encoded opaque data structure representing the user's Access Manager credential.

Table 14. IV header field descriptions (continued)

IV Header Field	Description
iv-remote-address	The IP address of the client. This value could represent the IP address of a proxy server or a network address translator (NAT).

Enabling authentication using IV headers

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable authentication using IV headers, assign the reference **iv-headers** to the **authentication** parameter; that is:

```
[common-modules]
authentication = iv-headers
```

To enable IV headers for post-authorization processing, assign the **post-authzn** parameter the keyword value **iv-headers** in the **[common-modules]** stanza in the `pdwebpi.conf` configuration file. That is:

```
[common-modules]
post-authzn = iv-headers
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library name. Ensure that the entry for IV header authentication exists; that is:

```
[modules]
iv-headers = pdwpi-iv-headers-module
```

Configuring IV header parameters

IV header authentication parameters are configured in the **[iv-headers]** stanza of the `pdwebpi.conf` configuration file.

The **accept** parameter specifies the types of IV header that are accepted for performing IV header authentication. By default the plug-in accepts all types of IV header. The valid options are: all, iv-creds, iv-user, iv-user-l, iv-remote-address. To enter more than one header type, separate the values with a comma.

For example:

```
[iv-headers]
accept = iv-creds,iv-user
```

The **generate** parameter specifies the type of IV headers to generate when forwarding proxied requests. By default the plug-in generates all types of IV header when forwarding proxied requests. The valid options are: all, iv-creds, iv-user, iv-user-l, iv-remote-address. To enter more than one header type, separate the values with a comma.

Configuring the IV header authentication mechanism for iv-remote-address

When using **iv-remote-address** in the IV Header you will need to specify the shared library for mapping HTTP authentication header information. The **http-request** parameter specifies the shared library for mapping HTTP authentication header information.

- On UNIX, the file that provides the built-in mapping function is a shared library called `libpdwpi-http-cdas`.
- On Windows, the file that provides the built-in mapping function is a DLL called `pdwpi-http-cdas`.

Table 15. IV Header shared library authentication mechanisms

Authn Mechanisms	Shared Library		
	Solaris Operating Environment	**AIX**	**Windows**
http-request	`libpdwpi-http-cdas.so`	`libpdwpi-http-cdas.a`	`pdwpi-http-cdas.dll`

You can configure the HTTP header authentication mechanism by entering the http-request parameter with the platform-specific name of the shared library file in the **[authentication-mechanisms]** stanza of the `pdwebpi.conf` configuration file, that is:

Solaris Operating Environment:

```
[authentication-mechanisms]
http-request = libpdwpi-http-cdas.so
```

Windows:

```
[authentication-mechanisms]
http-request = pdwpi-http-cdas.dll
```

Configuring HTTP header authentication

Access Manager supports authentication via custom HTTP header information supplied by the client or a proxy agent.

This mechanism requires a mapping function (a shared library) that maps the trusted (pre-authenticated) header data to a Access Manager identity. The plug-in can take this identity and create a credential for the user.

The plug-in assumes custom HTTP header data has been previously authenticated. For this reason, it is recommended that you implement this method exclusively, with no other authentication methods enabled. It is possible to impersonate custom HTTP header data.

By default, this shared library is built to map data from Entrust Proxy headers.

Enabling authentication using HTTP headers

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable authentication using HTTP headers, assign the reference 'http-hdr' to the **authentication** parameter; that is:

```
[common-modules]
authentication = http-hdr
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library name. Ensure that the entry for HTTP header authentication exists; that is:

```
[modules]
http-hdr = pdwpi-httphdr-module
```

Specifying header types

You must specify all supported HTTP header types in the **[http-hdr]** stanza of the `pdwebpi.conf` configuration file.

```
[http-hdr]
header = header_type
```

A standard configuration of HTTP headers permits only one header to be specified. To specify multiple HTTP headers, multiple instances of the HTTP header module must be configured.

For example:

```
[modules]
entrust-client-header = libpdwpi-http-header.so
some-other-header     = libpdwpi-http-header.so

[entrust-client-header]
header = entrust-client

[some-other-header]
header = some-other
```

Configuring the HTTP header authentication mechanism

The **http-request** parameter specifies the shared library for mapping HTTP authentication header information.

- On UNIX, the file that provides the built-in mapping function is a shared library called `libpdwpi-http-cdas`.
- On Windows, the file that provides the built-in mapping function is a DLL called `pdwpi-http-cdas`.

Table 16. HTTP header shared library authentication mechanisms

Authn Mechanisms	Shared Library		
	Solaris Operating Environment	**AIX**	**Windows**
http-request	`libpdwpi-http-cdas.so`	`libpdwpi-http-cdas.a`	`pdwpi-http-cdas.dll`

By default, this built-in shared library is hard-coded to map Entrust Proxy header data to a valid Access Manager identity. You must customize this file to authenticate other types of special header data and, optionally, map this data to an Access Manager identity. Refer to the *IBM Tivoli Access Manager WebSEAL Developer Reference* for API resources.

You can configure the HTTP header authentication mechanism by entering the http-request parameter with the platform-specific name of the shared library file in the **[authentication-mechanisms]** stanza of the `pdwebpi.conf` configuration file.

For example:

Solaris Operating Environment:

```
[authentication-mechanisms]
http-request = libpdwpi-http-cdas.so
```

Windows:

```
[authentication-mechanisms]
http-request = pdwpi-http-cdas.dll
```

Configuring IP address authentication

The IP Address of incoming requests can be used to both maintain session state and to authenticate client requests using values in the client address headers.

Configuring the plug-in to use the IP Address for maintaining session state is invalid without also configuring it to use the IP address to authenticate the client

request. However, usage of the IP Address to authenticate users is valid if the plug-in does not use the IP Address to track user sessions.

Enabling authentication using the IP address

The **[common-modules]** stanza in the pdwebpi.conf configuration file defines the use of all authentication methods. To enable authentication using the IP Address of the request initiator, assign the reference 'ip-addr' to the **authentication** parameter as in the following:

```
[common-modules]
authentication = ip-addr
```

To enable the use of the IP Address to track user sessions, assign the reference 'ip-addr' to the **session** parameter as in the following:

```
[common-modules]
session = ip-addr
```

The **[modules]** stanza in the pdwebpi.conf configuration file defines all available authentication mechanisms and their associated shared library names. Ensure that the entry for IP Address authentication exists, as in the following:

```
[modules]
ip-addr = pdwpi-ipaddr-module
```

Configuring the IP address authentication mechanism

The IP Address Authentication Mechanism is the same as that for HTTP Headers. The **http-request** parameter specifies the shared library for the IP Address authentication mechanism.

- On UNIX, the file that provides the built-in mapping function is a shared library called libpdwpi-http-cdas.
- On Windows, the file that provides the built-in mapping function is a DLL called pdwpi-http-cdas.

Table 17. IP address shared library authentication mechanisms

Authn Mechanisms	Shared Library		
	Solaris Operating Environment	**AIX**	**Windows**
http-request	libpdwpi-http-cdas.so	libpdwpi-http-cdas.a	pdwpi-http-cdas.dll

You can configure the IP address authentication mechanism by entering the http-request parameter with the platform-specific name of the shared library file in the **[authentication-mechanisms]** stanza of the pdwebpi.conf configuration file.

For example:

Solaris Operating Environment:

```
[authentication-mechanisms]
http-request = libpdwpi-http-cdas.so
```

Windows:

```
[authentication-mechanisms]
http-request = pdwpi-http-cdas.dll
```

Configuring tag value post-authorization processing

Often customers may wish to attach user specific information from LDAP (e.g. telephone number, e-mail address) to the headers of HTTP authenticated requests.

This allows multiple applications to access the attached information without having to constantly query the LDAP server. The nature of this information is that it is relatively static and is never updated by any of the applications that use it. This data is placed into the user credential as part of the **ivauthn** authentication process. This information can also be attached to the users credential through a user-implemented CDAS authentication module.

Enabling tag value processing

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable processing using tag values, assign the reference 'tag-value' to the **post-authzn** parameter; as in the following:

```
[common-modules]
post-authzn = tag-value
```

The **[modules]** stanza in the `pdwebpi.conf` configuration file defines all available authentication mechanisms and their associated shared library names. Ensure that the entry for tag value exists, as in the following:

```
[modules]
tag-value = pdwpi-tag-value-module
```

Configuring tag value parameters

Tag value parameters are configured in the **[tag-value]** stanza of the `pdwebpi.conf` configuration file.

```
[tag-value]
cache-definitions = yes
cache-refresh-interval = 60
```

The **cache-definitions** parameter enables or disables caching of tag value definitions that are attached to the object space. The **cache-refresh-interval** defines the refresh interval in seconds for the cache of definitions.

Setting authentication parameters for virtual hosts

The following example sets up a virtual host called foo.com that uses SSL session IDs where it can, BA headers where it can't use an SSL ID and has BA headers, and uses session cookies as a last resort to maintain session information. It supports certificate authentication ahead of basic authentication and on successful authentication adds an LTPA cookie to request to be handled by the web server. The example only shows the parameters defined here.

```
[pdweb-plugins]
virtual-host = foo.com

[modules]
ssl-id = libpdwpi-ssl-id.so
session-cookie = libpdwpi-session-cookie.so
ba = libpdwpi-ba.so
cert = libpdwpi-cert.so
ltpa = libpdwpi-ltpa.so

[foo.com]
session = ssl-id
session = ba
session = session-cookie

authenitcation = cert
authentication = ba

post-authzn = ltpa
```

Authentication methods can be configured on a per-module basis so that different modules can be shared between virtual hosts; as in the following:

```
[virtual_host_stanza]
# Optional modules stanza name to allow sharing of module
# configurations between virtual hosts
modules = new-modules-stanza

[new-modules-stanza]
# Order sensitive session module list - first one has highest priority
session = session_module
session = session_module
...
# Order sensitive authentication module list - first one has highest
# priority
authentication = authentication_module
authentication = authentication_module
...

# Order sensitive post-authorization module list - first one has
# highest priority
post-authzn = post_authorization_module
post-authzn = post_authorization_module
...
```

Supporting Multiplexing Proxy Agents (MPA)

Access Manager provides solutions for securing networks that use a Multiplexing Proxy Agent (MPA). Multiplexing Proxy Agents (MPA) are gateways that accommodate multiple client access. Gateways establish a single authenticated channel to the origin server and ″tunnel″ all client requests and responses through this channel. To the plug-in, the information across this channel initially appears as multiple requests from one client. The plug-in must distinguish between the authentication of the MPA server and the additional authentication of each individual client. A common example of such gateways are Wireless Access Protocol (WAP) gateways. Access Manager WebSEAL also acts as an MPA when configured with a junction to the host Web server to allow single signon between WebSEAL and the plug-in. To configure such a solution, the iv-header authentication module can be used. SeeChapter 6, “Web single signon solutions” on page 77 for more details on configuring for SSO.

Valid session data types and authentication methods

Since Access Manager Plug-in for Web Servers maintains an authenticated session for the MPA, it must simultaneously maintain separate sessions for each client. Therefore, the session data and authentication method used for the MPA must be different than the session data and authentication method used by the client. The table below lists the valid session types for the MPA and the client:

Table 18. Valid session data types for MPA

Valid Session Types	
MPA-to-plug-in	**Client-to-plug-in**
SSL Session Id	
HTTP Header	HTTP Header
BA Header	BA Header
IP Address	
Cookie	Cookie

- The client cannot use an SSL session ID as the session data type.

- As an example, if the MPA uses a BA header for the session data type, the client's choices for session data type include only HTTP header and cookie.
- If the MPA uses a HTTP header for session data, the client can use a different HTTP header type.
- The server-specific cookie contains session information only; it does not contain identity information.
- If MPA support is enabled, the use of SSL session IDs to maintain session state changes. Normally, having SSL session IDs configured to maintain session state, only the SSL session ID is used to maintain sessions for HTTPS clients. To allow the MPA to maintain a session with an SSL session ID and have clients maintain sessions using another method, this restriction is removed.

The authentication method used by the MPA to plug-in must be different than the authentication method used by the client to plug-in. The table below lists the valid authentication methods for the MPA and the client:

Table 19. Valid MPA authentication types

Valid Authentication Types	
MPA-to-plug-in	**Client-to-plug-in**
Basic Authentication	Basic Authentication
Forms	Forms
Token	Token
HTTP Header	HTTP Header
Certificate	
IP Address	

- As an example, if the MPA uses Basic Authentication, the client's choices for authentication methods includes Forms, token, and HTTP header.
- Certificates and IP address authentication methods are not valid for use by the client.
- Normally, if either Forms (or token) authentication is enabled for a particular transport, Basic Authentication is automatically disabled for that transport. If MPA support is enabled, this restriction is removed. This allows the MPA to log on, for example, with Forms (or token) and clients to log on with Basic Authentication over the same transport.

Authentication process flow for MPA and multiple clients

1. Make the following configuration changes:
   - Enable support for Multiplexing Proxy Agents in the configuration file.
   - Create an Access Manager account for the specific MPA gateway.
   - Grant Proxy ([PDWebPI]p) access for this account to the MPA Protected Object for the virtual host to which proxied requests will be directed. In the default configuration, this can be achieved by making the user a member of the **pdwebpi-mpa-servers** group.
2. Clients connect to the MPA gateway.
3. The gateway translates the request to a HTTP request.
4. The gateway authenticates the client.
5. The gateway establishes a connection with the plug-in using the client request.
6. The MPA authenticates to the plug-in (using a method distinct from the client) and an identity is derived for the MPA (which already has a plug-in account).

7. The plug-in verifies the MPA's membership in the **pdwebpi-mpa-servers** group.
8. A credential is built for the MPA and flagged as a special MPA type in the cache.

   Although this MPA credential accompanies each future client request, it is not used for authorization checks on these requests.
9. Now the plug-in needs to further identify the owner of the request.

   The MPA is able to distinguish the multiple clients for proper routing of logon prompts.
10. The client logs in and authenticates using a method distinct from the authentication type used for the MPA.
11. The plug-in builds a credential from the client authentication data.
12. The session data type used by each client must be distinct from the session data type used by the MPA.
13. The Authorization Server permits or denies access to protected objects based on the user credential and the object's ACL permissions.

Enabling MPA authentication

The **mpa-enabled** parameter in the **[pdweb-plugins]** stanza of the `pdwebpi.conf` configuration file enables or disables MPA authentication. The valid settings are **true** and **false** for enabling and disabling MPA authentication respectively. MPA authentication is disabled by default. MPA authentication can be set for individual virtual hosts by specifying the **mpa-enabled** parameter in the [*virtual_host*] stanza of the configuration file.

To identify a new session as being the primary session established by an MPA, an authorization decision is made testing for the Proxy ([PDWebPI]p) permission on the MPA protected object. By default the MPA protected object is defined to be /PDWebPI. To override this default setting, for example to define different sets of principals as representing MPAs for each virtual host, a value can be specified for the **mpa-protected-object** configuration parameter. This parameter may be overridden for each virtual host by specifying a value for it in the [*virtual_host*] stanza of the configuration file. For example to enable MPA access for the foo.com virtual host but not the bar.com virtual host use the following settings in the `pdwebpi.conf` configuration file:

```
[pdmweb-plugins]
virtual-host = foo.com
virtual-host = bar.com

[foo.com]
mpa-enabled = yes
```

To define members of the foo-mpa-servers group as being MPAs for requests to the foo.com virtual host and bar-mpa-servers group as being MPAs for requests to the bar.com virtual host, use the following configuration:

```
[pdweb-plugins]
virtual-host = foo.com
virtual-host = bar.com

[foo.com]
mpa-enabled = yes
mpa-protected-object = /PDWebPI/foo.com

[bar.com]
mpa-enabled = yes
mpa-protected-object = /PDWebPI/bar.com
```

and define the following Access Manager policy:

```
pdadmin> acl create foo-mpa
pdadmin> acl modify foo-mpa set group foo-mpa-servers T[PDWebPI]p
pdadmin> acl create bar-mpa
pdadmin> acl modify bar-mpa set group bar-mpa-servers T[PDWebPI]p
pdadmin> acl attach /PDWebPI/foo.com foo-mpa
pdadmin> acl attach /PDWebPI/bar.com bar-mpa
```

The **mpa-protected-object** configuration parameter specifies the object against which the authorization decision will be made.

Create a user account for the MPA

Refer to the *IBM Tivoli SecureWay Access Manager Base Administration Guide* and the *IBM Tivoli SecureWay Access Manager Web Portal Manager Administration Guide* for information on creating user accounts.

Add the MPA account to the pdwebpi-mpa-servers group

Access Manager Plug-in for Web Servers creates a group for easily administering MPA servers. This group is called **pdwebpi-mpa-servers**. The default-pdwebpi ACL attached to **/PDWebPI** grants Proxy ([PDWebPI]p) permission to members of the **pdwebpi-mpa-servers** group. When installed in an Access Manager secure domain that has at least one WebSEAL configured, the default-pdwebpi ACL is configured so that it also grants Proxy permission to members of the **webseal-servers** and **webseal-mpa-servers** groups. You may choose your own groups and ACLs used to control identification of principals as Multiplexing Proxy Agents.

Refer to the *IBM Tivoli SecureWay Access Manager Base Administration Guide* and the *IBM Tivoli SecureWay Access Manager Web Portal Manager Administration Guide* for information on managing groups.

Chapter 5. IBM Tivoli Access Manager Plug-in for Web Servers security policy

This chapter contains information that describes how you can configure and customize IBM Tivoli Access Manager (Access Manager) Plug-in for Web Servers security policy.

Topic Index:

- "Plug-in specific Access Control List (ACL) policies"
- "Three strikes logon policy" on page 65
- "Password strength policy" on page 66
- "Authentication strength Protected Object Policy (Step-up)" on page 68
- "Reauthentication Protected Object Policy" on page 71
- "Network-based authentication Protected Object Policy" on page 73
- "Quality of protection Protected Object Policy" on page 74
- "Handling unauthenticated users (HTTP / HTTPS)" on page 75

Plug-in specific Access Control List (ACL) policies

The following security considerations apply for the `/PDWebPI` container in the protected object space:

- The Access Manager Plug-in for Web Servers object begins the chain of ACL inheritance for the plug-in region of the object space.
- If you do not apply any other explicit ACLs, this object defines (through inheritance) the security policy for the entire Web space.
- The traverse permission is required for access to this object and any object below this point.

Refer to the *IBM Tivoli Access Manager Base Administrator's Guide* for complete information about Access Manager ACL policies.

/PDWebPI*host* or *virtual_host*

This subtree contains the object space of a particular plug-in instance. The following security considerations apply for this object:

- The traverse permission is required for access to any object below this point.
- If you do not apply any other explicit ACLs, this object defines (through inheritance) the security policy for the entire object space on this machine.

Plug-in ACL permissions

The following table describes the ACL permissions applicable for the Access Manager Plug-in for Web Servers region of the object space:

© Copyright IBM Corp. 2002

Table 20. Plug-in ACL permissions

Permission	Operation	Description
[PDWebPI]**r**	read	View any element other than a directory. Any HTTP GET or POST request requires this permission. There is no specific ″list″ permission for requesting a directory listing (A GET of a URL ending in /) – this is also checked with the [PDWebPI]r permission.
[PDWebPI]**d**	delete	Remove the Web object from the Web space. HTTP DELETE commands require this permission.
[PDWebPI]**m**	modify	Place / publish a HTTP object in the plug-in object space. A HTTP PUT request requires this permission.
T	traverse	Required for access to any object below this point

The plug-in also supports WebDAV operations as shown below.

Table 21. Plug-in WebDAV permissions

Task	Permission Required
PROPFIND	[PDWebPI]**R**
PROPPATCH	[PDWebPI]**M**
MKCOL	[PDWebPI]**N**

WebDAV operations are authorized based on the request URI – not on individual members of a collection. In addition, some other WebDAV operations are partially supported:

- **COPY** - Requires [PDWebPI]**R** on the collection so that the 'copy from' can be read. Permissions for the destination are not checked.
- **MOVE** - This is considered a copy then a delete. Requires [PDWebPI]**Rd** on the collection that you are moving from. Permissions for the destination are not checked.

Default /PDWebPI ACL policy

Core entries for the Access Manager Plug-in for Web Servers ACL, **default-pdwebpi**, include:

```
Group iv-admin              Tcmdbva[PDWebPI]rmdNRM

User sec_master             Tcmdbva[PDWebPI]rmdNRM

Any-other                   T[PDWebPI]rmdNRM

Unauthenticated             T[PDWebPI]rmdNRM
```

At installation, this default ACL is attached to the `/PDWebPI` container object in the object space.

The traverse permission allows expansion of the Web space as represented in the Web Portal Manager. The list permission allows the Web Portal Manager to display the contents of the Web space.

Three strikes logon policy

The three strikes logon policy, available for LDAP-based Access Manager installations, enables you to prevent computer password attacks by specifying a maximum number of failed log on attempts and a penalty lockout time. The policy creates a condition where a user must wait a period of time before making more log on attempts that fail. For example, a policy could dictate 3 failed attempts followed by a 180 second penalty. This type of logon policy can prevent random computer-generated log on attempts occurring many times a second.

The three strikes logon policy requires the joint contribution of two **pdadmin** policy command settings:

- Maximum number of failed log on attempts
  policy set max-login-failures
- Penalty for exceeding failed log on attempt setting
  policy set disable-time-interval
  The penalty setting can include an account lockout time interval or a complete disabling of the account.

If a logon policy is set (as an example) for three failed attempts followed by specific lockout time penalty, a fourth attempt (correct or incorrect) will result in an error page that states the account is temporarily unavailable because of password policy.

The time interval is specified in seconds - the minimum recommended time interval is 60 seconds.

If the **disable-time-interval** policy is set to ″disable″, the user is locked out of the account and the LDAP **account valid** attribute for this user is set to ″no″. An administrator re-enables the account through the Web Portal Manager.

Note: Setting the **disable-time-interval** to ″disable″ results in additional administration overhead. You may observe delays in replicating **account valid** information to the plug-in. This situation depends on your LDAP environment. In addition, certain LDAP implementations might experience performance degradation as a result of the **account valid** update operation. For these reasons it is recommended that you use a timeout interval.

Command syntax

The following **pdadmin** commands are appropriate only for use with an LDAP registry.

Table 22. pdadmin LDAP logon policy commands

Command	Description
policy set max-login-failures {*number* | **unset**} **[-user** *username***]**	
policy get max-login-failures [-user *username***]**	

Table 22. pdadmin LDAP logon policy commands (continued)

Command	Description
	Manages the policy controlling the maximum number of failed log on attempts allowed before a penalty is imposed. This command depends on a penalty set in the policy set disable-time-interval command. As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the LDAP registry. The default setting is 10 attempts.
policy set disable-time-interval {*number*|**unset**|**disable**} **[-user** *username***]**	
policy get disable-time-interval [-user *username***]**	
	Manages the penalty policy controlling the time period an account should be disabled if the maximum number of failed log on attempts is reached. As the administrator, you can apply this penalty policy to a specific user or apply the policy globally to all users listed in the LDAP registry. The default setting is 180 seconds.

Password strength policy

An Access Manager LDAP-based installation provides two means of controlling the construction of passwords:

- Five **pdadmin** password policy commands
- A plugable authentication module (PAM) that allows you to customize a password policy

  Refer to the Access Manager Authorization C API Developer's Reference

Password strength policy set by the pdadmin utility

The five password strength attributes implemented through the **pdadmin** utility include:

- Minimum password length
- Minimum alphabetic characters
- Minimum non-alphabetic characters
- Maximum repeated characters
- Spaces allowed

These policies are enforced when you create a user with **pdadmin** or the Web Portal Manager, and when a password is changed with **pdadmin**, the Web Portal Manager, or the **pkmspasswd** utility.

Command syntax

The following pdadmin commands are only appropriate for use with an LDAP registry. The unset option disables this policy attribute – that is, the policy is not enforced.

Table 23. pdadmin LDAP password strength commands

Command	Description
policy set min-password-length {*number*|**unset**} **[-user** *username***]**	
policy get min-password-length [-user *username***]**	
	Manages the policy controlling the minimum length of a password. As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the default registry. The default setting is 8.
policy set min-password-alphas {*number*|**unset**} **[-user** *username***]**	
policy get min-password-alphas [-user *username***]**	
	Manages the policy controlling the minimum number of alphabetic characters allowed in a password. As the administrator, you can apply this penalty policy to a specific user or apply the policy globally to all users listed in the default registry. The default setting is 4.
policy set min-password-non-alphas {*number*|**unset**} **[-user** *username***]**	
policy get min-password-non-alphas [-user *username***]**	
	Manages the policy controlling minimum number of non-alphabetic (numeric) characters allowed in a password. As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the default registry. The default setting is 1.
policy set max-password-repeated-chars {*number*|**unset**} **[-user** *username***]**	
policy get max-password-repeated-chars [-user *username***]**	
	Manages the policy controlling the maximum number of repeated characters allowed in a password. As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the default registry. The default setting is 2.
policy set password-spaces {yes|no|unset} [-user *username***]**	
policy get password-spaces [-user *username***]**	
	Manages the policy controlling whether a password can contain spaces. As the administrator, you can apply this policy to a specific user or apply the policy globally to all users listed in the default registry. The default setting is unset.

Valid and invalid password examples: The following table illustrates several password examples and the policy results based on the default values of the five pdadmin parameters:

Table 24. Password examples

Example	Result
password	Not valid: must contain at least one non-alphabetic character.
pass	Not valid: must contain at least 8 characters.
passs1234	Not valid: contains more than two repeated characters.
12345678	Not valid: must contain at least four alphabetic characters.
password3	Valid.

Specific user and global settings

The **pdadmin** policy commands can be set for a specific user (with the - user option) or globally (by not using the - user option). Any user-specific setting overrides a global setting for the policy. You can also disable (unset) a policy parameter, which means the parameter contains no value. Any policy with the unset option is not checked or enforced.

For example:

```
pdadmin> policy set min-password-length 8
pdadmin> policy set min-password-length 4 -user matt
pdadmin> policy get min-password-length
Minimum password length: 8
pdadmin> policy get min-password-length -user matt
Minimum password length: 4
```

(User matt has a minimum password length policy of 4 characters; all other users have a minimum password length policy of 8.)

```
pdadmin> policy set min-password-length unset -user matt
```

(User matt is now governed by the global minimum password length policy of 8 characters.)

```
pdadmin> policy set min-password-length unset
```

(All users, including user matt now have no minimum password length policy.)

Authentication strength Protected Object Policy (Step-up)

The authentication strength Protected Object Policy (POP) makes it possible to control access to objects based on the authentication method that they use.

You can use this functionality – sometimes known as step-up authentication – to ensure that users accessing more sensitive resources use a stronger authentication mechanism. You might want this condition because of the greater threat of improper access.

For example, you can provide greater security to a region of the Web space by applying a step-up POP policy that requires a stronger level of authentication than the client used when initially entering the plug-in domain.

Step-up authentication can also be set for each specific virtual host on a Web server, allowing individual virtual hosts to carry their own step-up levels of authentication without being subject to server wide policy implementations.

Authentication strength policy is set in the IP Endpoint Authentication Method attribute of a POP policy.

Configuring levels for step-up authentication

The first step in configuring authentication-specific access is to configure the supported authentication methods and determine the order in which these authentication methods should be considered stronger. Refer to, Chapter 4, “IBM Tivoli Access Manager Plug-in for Web Servers authentication” on page 29 for details on configuring authentication mechanisms.

Any client accessing a Web server through the plug-in has an authentication level, such as ″unauthenticated″ or ″password″, which indicates the method by which the client last authenticated through the plug-in.

In some situations it may be necessary to enforce minimum ″safe″ levels of authentication required to access certain Web space objects. For example, in one environment, authentication by token passcode may be considered more secure than authentication by username and password. Another environment could have different standards.

Rather than forcing clients to restart their sessions when they do not meet the required level of authentication, the step-up authentication mechanism gives clients a second chance to re-authenticate using the required method (level).

Step-up authentication means that a user is not immediately shown a ″denied″ message when they try to access a resource that requires a ″higher″ authentication level than the one they logged on with. Instead, they are presented with a new authentication prompt that requests information to support the higher authentication level. If they are able to supply this level of authentication then their original request will be permitted.

Access Manager Plug-in for Web Servers recognizes a variety of authentication methods (levels) for use in the step-up authentication mechanism:

- unauthenticated
- forms
- IP address
- HTTP Header
- token
- certificate
- IV headers
- failover cookie

You configure authentication levels in the **[authentication-levels]** or **[authentication-levels:***virtual_host_label***]** stanza of the `pdwebpi.conf` configuration file. For example:

```
[authentication-levels]
1 = BA
2 = iv-headers
3 = cert
```

Based on the order of the methods in the list, each method is assigned a level index.

- Unauthenticated is assumed to have a level of 0.
- Subsequent methods can be placed in any order. See "Step-up authentication notes and limitations" on page 71
- By default, Basic Authentication is configured as level 1.
- There must be at least two entries to enable step-up authentication.
- Levels for authentication mechanisms can be set for specific virtual hosts by specifying the levels using a stanza with the form: **[authentication-levels:***virtual_host_name***]**

Note: See Chapter 4, "IBM Tivoli Access Manager Plug-in for Web Servers authentication" on page 29 for detailed information about setting up the required authentication mechanisms.

Enabling step-up authentication

Step-up authentication is implemented via a POP policy placed on the objects requiring authentication sensitive authorization. You use the IP Endpoint Authentication Method attribute of a POP policy.

The **pdadmin pop modify set ipauth** command specifies both the allowed networks and the required authentication level in the **IP Endpoint Authentication Method** attribute.

The configured authentication levels can be linked to IP address ranges. This method is intended to provide management flexibility. If filtering users by IP address is not important, you can set a single entry for **anyothernw** (any other network). This setting will affect all accessing users, regardless of IP address, and require them to authenticate at the specified level. This is the most common method for implementing step-up authentication.

Syntax:

```
pdadmin> pop modify pop_name set ipauth anyothernw level_index
```

The **anyothernw** entry is used as a network range that will match any network not otherwise specified in the POP. This method is used to create a default entry which could either deny all unmatched IP addresses or allow anyone access who can meet the authentication level requirement.

By default, **anyothernw** appears in a POP with an authentication level index of 0. The entry appears as ″Any Other Network″ in the **pop show** command:

```
pdadmin> pop show test
  Protected object policy:  test
        Description:  Test POP
        Warning:  no
        Audit level:  none
        Quality of protection:  none
        Time of day access: sun, mon, tue, wed, thu, fri, sat:
            anytime:local
        IP Endpoint Authentication Method Policy
            Any Other Network 0
```

Example

1. Configure authentication levels in pdwebpi.conf:

   ```
   [authentication-levels] or [authentication-levels:virtual_host_label]
   1 = BA
   2 = token
   ```

2. Configure IP Endpoint Authentication Method POP attribute:

   ```
   pdadmin> pop modify test set ipauth anyothernw 2

   pdadmin> pop show test
           Protected object policy:  test
           Description:  Test POP
           Warning:  no
           Audit level:  none
           Quality of protection:  none
           Time of day access: mon, wed, fri:anytime:local
           IP Endpoint Authentication Method Policy
               Any Other Network 2
   ```

   Therefore, users accessing objects protected by the test POP require level 2 authentication or will be forced to authenticate with the token method.
   See also "Network-based authentication Protected Object Policy" on page 73.

Step-up authentication notes and limitations

1. Step-up authentication is supported over both HTTP and HTTPS.
2. You cannot step-up from the HTTP protocol to HTTPS.
3. Authentication methods not specified in the **[authentication-levels]** stanza default to level 1.
4. Authentication methods can only be specified once in the level list.
5. Incorrect configuration of step-up authentication levels results in the disabling of step-up functionality within the plug-in. This situation can lead to unexpected authentication behavior, such as the password logon page being issued for objects protected by a POP that requires the token passcode authentication method.

   After configuring step-up authentication mechanisms, check the pdwebpi.log file for reports of any configuration errors.

Reauthentication Protected Object Policy

Access Manager Plug-in for Web Servers can force a user to perform an additional log on (reauthentication) to ensure that a user accessing a protected resource is the same person who initially authenticated at the start of the session. Reauthentication can be activated by a Protected Object Policy (POP) on the protected object or by expiration of the session cache inactivity timeout value. This section discusses reauthentication based on security policy as dictated by a POP extended attribute. Refer to "Configuring the plug-in session / credentials cache" on page 38 for details on configuring the Session / Credential Cache.

Conditions affecting POP reauthentication

Forced reauthentication provides additional protection for sensitive resources in the secure domain. Reauthentication based on security policy is activated by a specific extended attribute in a POP that protects the requested resource object. The POP can be directly attached to the object, or the object can inherit the POP conditions from a parent object. Reauthentication is supported by the following plug-in authentication methods:

- Forms (user name and password) authentication
- Token authentication

In addition, a custom user name/password CDAS can be written to support reauthentication.

Reauthentication assumes the user has initially logged in to the secure domain and that a valid credential exists for the user. During reauthentication, the user must log on using the same identity that generated the existing credential. Access Manager preserves the user's original session information, including the credential, during reauthentication. The credential is not replaced during reauthentication.

During reauthentication, the plug-in also caches the request that prompted the reauthentication. Upon successful reauthentication, the cached data is used to rebuild the request.

If reauthentication fails, the plug-in returns the logon prompt again. If reauthentication succeeds, but the ACL check fails for that resource, a 403 ″Forbidden″ is returned and the user is denied access to the requested resource. In either case, the user is never logged off. Using a still valid credential, the user can abort the reauthentication process (by requesting another URL) and still participate in the secure domain by accessing other resources that do not require reauthentication.

Configuration is available to reset the plug-in session cache lifetime timer. In addition, a grace period can be configured to allow sufficient time for the reauthentication process to complete before the session cache lifetime timeout expires. Refer to, "Configuring the plug-in session/credentials cache" on page 38 for details.

Creating and applying the reauthentication POP

Forced reauthentication based on security policy is configured by creating a protected object policy (POP) with a special extended attribute named ″reauth″. You can attach this POP to any object that requires the extra protection provided by forced reauthentication.

Remember that all children of the object with the POP also inherit the POP conditions. Each requested child object requires a separate reauthentication.

Use the **pdadmin pop create**, **pdadmin pop modify**, and **pdadmin pop attach** commands. The following example illustrates creating a POP called ″secure″ with the reauth extended attribute and attaching it to an object:

```
pdadmin>pop create secure
pdadmin>pop modify secure set attribute reauth true
pdadmin>pop attach /PDWebPI/hostA/budget.html secure
```

Anyone attempting to access budget.html is forced to reauthenticate using the same identity and authentication method that generated the existing credential.

If the user requesting the resource is unauthenticated, the POP forces the user to authenticate. Reauthentication is required for every access to objects protected by a reauthentication policy.

In situations when most objects in a directory require reauthentication though a few do not, it is best to attach a POP to the entire directory including the ″reauth″

extended attribute. For those objects that do not require reauthentication, attach a POP that is identical to that for the directory though excluding the ″reauth″ extended attribute.

Details about the **pdadmin** command line utility can be found in the *IBM Tivoli Access Manager Base Administrator's Guide.*

Network-based authentication Protected Object Policy

The network-based authentication Protected Object Policy (POP) makes it possible to control access to objects based on the IP address of the user. You can use this functionality to prevent specific IP addresses (or IP address ranges) from accessing any resources in your secure domain.

You can also apply step-up authentication configuration to this policy and require a specific authentication method for each specified IP address range.

Network-based authentication policy is set in the IP Endpoint Authentication Method attribute of a POP policy. You must specify two requirements in this attribute:

- Authentication levels
- Allowed networks

For details on specifying configuration levels, refer to "Configuring levels for step-up authentication" on page 69

Specifying IP addresses and ranges

After configuring the authentication levels you must specify the IP addresses and IP address ranges permitted by this POP policy.

The **pdadmin pop modify set ipauth add** command specifies both the network (or network range) and the required authentication level in the IP Endpoint Authentication Method attribute.

Syntax:

```
pdadmin> pop modify pop_name set ipauth add network netmask level_index
```

The configured authentication levels are linked to IP address ranges. This method is intended to provide flexibility. If filtering users by IP address is not important, you can set a single entry for **anyothernw** (any other network). This setting affects all accessing users, regardless of IP address, and require them to authenticate at the specified level.

Syntax:

```
pdadmin> pop modify pop_name set ipauth anyothernw level_index
```

Conversely, if you wish to ignore the authentication level and only want to allow or deny access based on IP address, you can use level 0 for ranges that you want to allow in and ″forbidden″ for ranges you want to deny.

The **anyothernw** entry is used as a network range that matches any network not otherwise specified in the POP. This method used to create a default entry which could either deny all unmatched IP addresses or allow anyone access who meet the authentication level requirement.

By default, **anyothernw** appears in a POP with an authentication level index of 0. The entry appears as ″Any Other Network″ in the **pop show** command:

```
pdadmin> pop show test
        Protected object policy:  test
        Description:  Test POP
        Warning:  no
        Audit level:  none
        Quality of protection:  none
        Time of day access: sun, mon, tue, wed, thu, fri, sat:
            anytime:local
        IP Endpoint Authentication Method Policy
            Any Other Network 0
```

Refer to, “Configuring levels for step-up authentication” on page 69 for a more detailed discussion on setting authentication levels.

Example

Require users from IP address range 9.0.0.0 and netmask 255.0.0.0 to use level 1 authentication (″password″ by default):

```
pdadmin> pop modify test set ipauth add 9.0.0.0 255.0.0.0 1
```

Require a specific user to use level 0 authentication:

```
pdadmin> pop modify test set ipauth add 9.1.2.3 255.255.255.255 0
```

Prevent all users (other than those specified as in the examples above) from accessing the object:

```
pdadmin> pop modify test set ipauth anyothernw forbidden
```

Disabling step-up authentication by IP address

Syntax:

```
pdadmin> pop modify pop_name set ipauth remove network netmask
```

For example:

```
pdadmin> pop modify test set ipauth remove 9.0.0.0 255.0.0.0
```

Network-based authentication algorithm

Access Manager Plug-in for Web Servers uses the following algorithm to process the conditions in a POP:

1. Check the IP endpoint authentication method policy on the POP.
2. Check ACL permissions.
3. Check time-of-day policy on the POP.
4. Check the audit level policy on the POP.

Quality of protection Protected Object Policy

The quality of protection Protected Object Policy (POP) attribute allows you to specify what level of data protection is required when performing an operation on an object.

```
pdadmin> pop modify pop_name set qop {none|integrity|privacy}
```

Table 25. QOP level descriptions

QOP level	Description
privacy	Data encryption is required (SSL).

Table 25. QOP level descriptions (continued)

QOP level	Description
integrity	Use some mechanism to ensure that the data has not changed.
none	No method of data protection is used.

For example:

```
pdadmin> pop modify test set qop privacy
```

The quality of protection POP attribute permits a single transaction when the ″yes″ response to the ACL decision also includes the required quality of protection level. If the plug-in cannot guarantee the required level of protection, the request is denied.

Handling unauthenticated users (HTTP/HTTPS)

Access Manager Plug-in for Web Servers accepts requests from both authenticated and unauthenticated users over HTTP and HTTPS. The plug-in then relies on the Authorization Server to enforce security policy by permitting or denying access to protected resources.

The following conditions apply to unauthenticated users who access over SSL:

- The exchange of information between the unauthenticated user and the plug-in is encrypted – as it is with an authenticated user.
- An SSL connection between an unauthenticated user and the plug-in only requires server-side authentication.

Processing a request from an anonymous client

1. An anonymous client makes a request to the web server through the plug-in (using HTTP or HTTPS).
2. The plug-in creates an unauthenticated credential for this client.
3. The request proceeds, with this credential, to the protected Web object.
4. The Authorization Server checks the permissions on the unauthenticated entry of the ACL for this object, and permits or denies the requested operation.
5. Successful access to this object depends on the unauthenticated ACL entry containing at least the read (r) and traverse (T) permissions.
6. If the request fails the authorization decision, the client receives a logon form (BA or Forms-based).

Forcing user log on

You can force an unauthenticated user to log on by correctly setting the appropriate permissions on the unauthenticated entry in the ACL policy that protects the requested object.

The read [PDWebPI]**r** and traverse (T) permissions allow unauthenticated access to an object.

To force an unauthenticated user to log on, remove the read [PDWebPI]**r** permission from the unauthenticated entry in the ACL policy that protects the object. The user receives a logon prompt (BA or Forms-based).

Applying unauthenticated HTTPS

There are many practical business reasons for supporting unauthenticated access to the plug-in enhanced Web server over HTTPS:

- Some applications do not require a personal log on, but require sensitive information, such as addresses and credit card numbers. Examples include online purchases of airline tickets and other merchandise.
- Some applications require that you register for an account with the business before you can proceed with further transactions. Again, sensitive information must be passed over the network.

Controlling unauthenticated users with ACL/POP policies

Note: The ″any-authenticated″ entry type is equivalent to the ″any-other″ entry type.

1. To permit unauthenticated user access to public objects, protect the public content with an ACL that contains at least the read [PDWebPI]**r** and traverse (T) permissions for the unauthenticated and any-authenticated entries:

   ```
   unauthenticated T[PDWebPI]r
   any-authenticated T[PDWebPI]r
   ```

   Note: The **unauthenticated** entry is a mask (a bitwise ″and″ operation) against the **any-authenticated** entry when permissions are determined. A permission for **unauthenticated** is granted only if the permission also appears in the **any-authenticated** entry. Since **unauthenticated** depends on **any-authenticated**, it makes little sense for an ACL to contain **unauthenticated** without **any-authenticated**. If an ACL does contain **unauthenticated** without **any-authenticated**, the default response is to grant no permissions to **unauthenticated**.
2. To require encryption (SSL), protect the content with a Protected Object Policy (POP) that specifies privacy as a condition.
   See “Quality of protection Protected Object Policy” on page 74.

Chapter 6. Web single signon solutions

When Access Manager Plug-in for Web Servers is implemented as an authorization service to provide protection to a secure domain, there is often a requirement to provide solutions for single signon to resources within that domain. This chapter discusses single signon solutions for the Web space protected by Access Manager Plug-in for Web Servers.

Topic Index:

- "Single signon concepts"
- "Automatically signing-on to a secured application"
- "Single signon to the plug-in from WebSEAL or other proxy" on page 80
- "Using the Failover cookie for single signon" on page 81

Single signon concepts

When a protected resource is located on the plug-in enhanced Web application server, a client requesting that resource can be required to perform multiple log ons when accessing different secure applications. Each log on is likely to require different logon identities.

The problem of administering and maintaining multiple logon identities can often be solved with a single signon (SSO) mechanism. SSO allows the user to access a resource using only one initial log on. Any further log on requirements for resources on the Web server are handled transparent to the user.

There are four main single signon architectures currently supported by Access Manager Plug-in for Web Servers. These are:

1. One plug-in instance providing single signon to more than one secure application on a server.
2. Single signon to the plug-in from WebSEAL or other proxy agent such as a WAP gateway.
3. Use of failover cookies to provide single signon between different domains.
4. e-Community single signon, where a user authenticates once and is issued a token that allows them to access other domains within a virtual community of domains without the need to re-authenticate.

The first three SSO scenarios are discussed in this chapter. The fourth scenario is the topic of the next chapter.

Automatically signing-on to a secured application

HTTP headers and LTPA cookies (when the application is WebSphere Application Server) can be used to achieve SSO to applications on a server that are protected by one plug-in instance.

After initial authentication of the client, the plug-in can build a HTTP header containing client identity information that can be used for automatic authentication to secure applications running on the server. In a similar way, an LTPA cookie can be used to achieve SSO to a Web application server such as WebSphere.

© Copyright IBM Corp. 2002

Configuring single signon to secure applications using HTTP headers

The HTTP headers used for signing on to an application are generated by the iv-headers post-authorization module. The set of headers that can be generated are collectively called IV headers.

After the successful authorization of a user request, the plug-in can insert IV headers that define the client's identity into the request for processing by the application. This header information can be used as proof of the user's identity when the request is handled by an application hosted by the secured Web server. The user is spared the need to log on each time a new secure application is accessed.

Configured for post-authorization processing, IV headers are inserted with one, some or all of the iv-user, iv-user-l, iv-creds, iv-groups, iv-remote-address HTTP header types. These header types are described in the following table.

Table 26. IV header field descriptions

IV Header Field	Description
iv-user	The short name of the Access Manager user. Defaults to unauthenticated if the client is unauthenticated (unknown).
iv-user-l	The full domain name of the user (long form). E.g. LDAP distinguished name.
iv-groups	A list of the groups to which the user belongs.
iv-creds	Encoded opaque data structure representing the user's Access Manager credential.
iv-remote-address	The IP address of the client. This value could represent the IP address of a proxy server or a network address translator (NAT).

Enabling and disabling generation of IV headers

To enable the plug-in to insert IV headers into authorized requests, the plug-in needs to be configured to use IV headers for post-authorization processing. The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable IV headers for post-authorization processing, assign the parameter **post-authzn** the keyword value **iv-headers** in the **[common-modules]** stanza in the `pdwebpi.conf` configuration file. That is:

```
[common-modules]
post-authzn = iv-headers
```

Configuring IV header parameters

IV header authentication parameters are configured in the **[iv-headers]** stanza of the `pdwebpi.conf` configuration file.

The **generate** parameter specifies the type of IV headers to generate when forwarding proxied requests. By default the plug-in generates all types of IV headers when forwarding proxied requests. The valid options are: all, iv-creds, iv-user, iv-user-l, iv-remote-address. To enter more than one header type, separate the values with a comma.

For example:

```
[iv-headers]
generate = iv-creds,iv-user,iv-user-l
```

Single signon to WebSphere application server using LTPA cookies

When the plug-in is installed as a protective layer on a WebSphere application server, accessing clients are faced with two potential logon points – the plug-in and secure applications served by WebSphere. To provide a single point of logon in this situation the plug-in can be configured to generate and pass the cookie-based lightweight third party authentication (LTPA) mechanism onto Web application servers that support LTPA cookies.

When a user makes a request for a resource on a server, the user must first authenticate to the plug-in. Upon successful authentication, the plug-in generates an LTPA cookie on behalf of the user. The LTPA cookie, which serves as an authentication token for the Web application server, contains user identity and password information. This information is encrypted using a password-protected secret key shared between the plug-in and the application server.

The plug-in inserts the cookie in the HTTP header of the request that is sent to the web application server. The application server receives the request, decrypts the cookie, and authenticates the user based on the identity information supplied in the cookie.

To improve performance, the plug-in stores the LTPA cookie in the session cache and uses the cached LTPA cookie for subsequent requests during the same user session. For details on setting the parameters for the session cache refer to, “Configuring the plug-in session/credentials cache” on page 38

Configuring single signon to WebSphere using LTPA cookies

Use of LTPA cookies to achieve single signon to application servers supporting LTPA cookies, is part of the plug-in’s post-authorization processing. To enable this functionality, enter the key value **ltpa** for the parameter **post-authzn** in the **[common-modules]** stanza of the `pdwebpi.conf` configuration file; that is:

```
[common-modules]
post-authzn = ltpa
```

LTPA cookie configuration is performed in the **[ltpa]** stanza of the `pdwebpi.conf` configuration file. The following parameters require configuration.

Table 27. LTPA configuration parameters.

Parameter	Description
ltpa-keyfile	The full path name of the key file used to encrypt the identity information contained in the cookie.
ltpa-stash-file	The location of the password stash file. If no password stash file exists, this entry should be commented out.
ltpa-password	The password to use when a password stash file does not exist.
ltpa-lifetime	The lifetime, in seconds, of the LTPA cookie.

Technical notes for LTPA single signon

- The key file contains information about a specific Web application server. If you add more than one application server to the same plug-in, all servers will share the same key file.
- For single signon to succeed, the plug-in and the application server must in some way share the same registry information.

- The application server is responsible for setting up LTPA and the creation of the shared secret key.

Single signon to the plug-in from WebSEAL or other proxy

When the plug-in enhanced Web server receives requests from a trusted application such as WebSEAL or a multi-plexing proxy agent, IV headers may be inserted into the requests relayed to the plug-in. IV headers contain information that identify the originating client rather than the relaying server. The information in the headers is used to construct an originating client credential for authorization purposes.

If the plug-in is configured to use IV Headers to perform client authentication, the plug-in creates a client credential using the identity extracted from an IV header found in the transaction request. Since it is easy for clients to fake IV headers, such a credential is only created when the 'use secondary authenticator' flag in the authenticate request is set.

For authentication, IV headers can be configured to accept one, some, or all of iv-user, iv-user-l, iv-creds or iv-remote-address headers in the request as proof of authentication when received via a proxy. The iv-remote-address header is used to record the real remote address of the user. These IV header types are recognized by Access Manager and WebSEAL.

Table 28. IV header field descriptions

IV Header Field	Description
iv-user	The short name of the client. Defaults to unauthenticated if the client is unauthenticated (unknown).
iv-user-l	The full domain name of the user (long form).
iv-groups	A list of the groups to which the client belongs.
iv-creds	Encoded opaque data structure representing an Access Manager credential.
iv-remote-address	The IP address of the client. This value could represent the IP address of a proxy server or a network address translator (NAT).

In order to be accepted as proof of client identity, WebSEAL or other proxy must itself be authenticated to the plug-in. This is typically achieved by a mutually authenticated SSL connection between the proxy and the Web server secured by the plug-in.

Configuring IV header single signon to Access Manager Plug-in for Web Servers

Enabling and disabling authentication using IV headers

The **[common-modules]** stanza in the `pdwebpi.conf` configuration file defines the use of all authentication methods. To enable authentication using IV headers, assign the reference 'iv-header' to the **authentication** parameter; that is:

```
[common-modules]
authentication = iv-header
```

Configuring IV header parameters

IV header authentication parameters are configured in the **[iv-headers]** stanza of the `pdwebpi.conf` configuration file.

The **accept** parameter specifies the types of IV header that are accepted for performing IV header authentication. By default the plug-in accepts all types of IV header. The valid options are: all, iv-creds, iv-user, iv-user-l, iv-remote-address. To enter more than one header type, separate the values with a comma.

For example:

```
[iv-headers]
accept = iv-creds,iv-user
```

Using the Failover cookie for single signon

With failover cookies configured for post-authorization processing, the plug-in encrypts a client's credential data in either a server specific or domain wide cookie. The cookie is placed on the browser when the client first connects. When the client attempts to access another secure server within the domain, the cookie is presented to the next server the client is redirected to. The cookie is used for automatic re-authentication so the client is spared the task of re-authenticating manually. The plug-ins on replicated servers share a common key that decrypts the credential information held in the cookie, establishing a new session.

Enabling single signon using Failover cookies

Failover cookies can be configured to perform authentication and post-authorization tasks.

Plug-ins configured for post-authorization processing using failover cookies, encrypt and store a credential as a failover cookie in the transaction response.

Plug-ins configured for using failover cookies for performing authentication, re-authenticate clients using the encrypted credential from a failover cookie found in the transaction request.

To enable SSO using failover cookies, the failover module must be configured for authentication and post-authorization. Assign the reference 'failover' to the **authentication** and **post-authzn** parameters in the **[common-modules]** stanza of the configuration file; that is:

```
[common-modules]
authentication = failover
post-authzn = failover
```

Note: Failover cookie authentication must be configured as the initial authentication method when other authentication mechanisms are configured along with failover cookies.

Configuring Failover cookie parameters

Failover cookie authentication parameters are configured in the **[failover]** stanza of the `pdwebpi.conf` configuration file.

The **failover-cookies-keyfile** parameter specifies the file that is used to encrypt and decrypt credential data in the failover cookie. For example:

```
[failover]
failover-cookies-keyfile = failover.key
```

The key file must be created using the program **pdwpi-cdsso-key-gen** located in the *install_path*`/bin` directory.

Usage:

```
./pdwpi-cdsso-key-gen key_file_name_to_create
```

The **failover-cookies-lifetime** parameter defines the valid lifetime of the failover-cookie in minutes. This is the time between cookie creation and the cookie being disabled. The default value is 30 minutes.

```
[failover]
failover-cookies-lifetime = 30
```

The **enable-failover-cookie-for-domain** parameter enables or disables the cookie validity within the entire domain. To achieve SSO to all servers within a domain, set this parameter to `true`.

For example:

```
 [failover]
enable-failover-cookie-for-domain = true
```

Chapter 7. e-Community single signon

When Access Manager Plug-in for Web Servers is implemented to provide protection to a secure domain, there is often a requirement to provide solutions for single signon to resources. This chapter discusses the plug-in e-community single signon solution.

Topic index:

- "Introducing e-community single signon"
- "e-Community single signon features and requirements" on page 84
- "e-Community single signon process flow" on page 84
- "Understanding the e-community cookie" on page 86
- "Understanding the ″Vouch For″ request and reply" on page 86
- "Understanding the ″Vouch For″ token" on page 87
- "Encrypting the ″Vouch For″ token" on page 87
- "Configuring an e-community" on page 88
- "Configuring e-community single signon - an example" on page 90

Introducing e-community single signon

Access Manager Plug-in for Web Servers e-community single signon functionality allows users to access resources across multiple servers in multiple domains without requiring re-authentication.

An ″e-community″ is a group of distinct domains (Access Manager or DNS) that participate in a business relationship. These participating domains can be configured as part of one business (and perhaps using different DNS names for geographic reasons), or as different businesses with a shared relationship (for example, company headquarters, a life insurance company, and a financial management company).

In either scenario, there is always one domain that is designated the ″home″ or ″owner″ domain. In the case of participating businesses, the home domain owns the business agreements that govern the e-community.

In both scenarios, authentication information about the users who participate in the e-community (including the user names and passwords used for authentication) is maintained in the home domain. This arrangement allows a single point of reference for administration issues, such as help desk calls within the e-community that all refer to the home domain.

Alternatively, you can use the Access Manager Web Portal Manager to delegate the management of this information such that participating domains have responsibility for the administration of their own users.

The home domain ″owns″ the users – that is, it controls the user's authentication information. Regardless of where a user makes a request for resources, the home domain is always where the user must authenticate.

© Copyright IBM Corp. 2002

Authentication occurs against a master authentication server (MAS) – a server (or set of replica servers) that is located in the home domain and configured to authenticate all users. The duty of the MAS should be restricted to providing authentication services. The MAS should not contain resources that are available to users.

Once a user has successfully authenticated to the MAS, the MAS generates a ″vouch for″ token. This token is passed back to the server where the user is making the request. The server treats this ″vouch for″ token as proof that the user has successfully authenticated to the MAS and can participate in the e-community.

The transfer of information between e-community domains is described in detail in the section “e-Community single signon process flow”

e-Community single signon features and requirements

- e-Community functionality supports access via direct URL’s (bookmarks) to resources.
- e-Community implementation requires a consistent configuration across all plug-ins in all domains participating in the e-community
- All users who are participating in the e-community authenticate against a single master authentication server (MAS) located in the home domain.
- The e-community implementation allows for ″local″ authentication in remote domains if the user does not have a valid account with the MAS.

  A user who fails authentication with the MAS when requesting a resource in a non-MAS (but participating) domain is given the option to authenticate to the local server where the request is being made.
- The MAS (and eventually other selected servers in the remote domains) ″vouch for″ the user’s authenticated identity.
- Domain-specific cookies are used to identify the server that can provide ″vouch for″ services. This allows servers in a remote domain to request ″vouch for″ information locally. The encrypted contents of e-community cookies do not contain user identity or security information.
- Special tokens are used to pass encrypted ″vouched for″ user identities. The ″vouch for″ token does not contain actual user authentication information. Integrity is provided by shared secret key (triple-DES). The token contains a time-out (lifetime) value to limit the duration of the token validity.
- The e-community implementation is supported on both HTTP and HTTPS.
- Individual e-community domains manage their own user identities and associated privileges. You can use the Cross Domain Mapping Function (CDMF) API to map a user from a remote domain to a valid user in the local domain.

  If the e-community domains share global user identities, this mapping function is not required.
- Configuration for e-community is set in the `pdwebpi.conf` file of each participating plug-in.

e-Community single signon process flow

An e-community consists of a plug-in enhanced master authentication server (MAS) and additional plug-in enhanced servers acting as an e-community. The e-community implementation is based on a ″vouch for″ system. Normally, when an unauthenticated user requests a resource through the plug-in they are prompted for authentication information. In an e-community configuration, the plug-in server

identifies a ″vouch for″ server and requests verification from this ″vouch for″ server that the user has authenticated. The ″vouch for″ server stores valid credential information for the user.

For the user's first request, the ″vouch for″ server is always the MAS. The MAS continues to serve as the ″vouch for″ server for resources located in the home domain. As the user continues with resource requests across the e-community, an individual server in each remote domain can build its own credential for the user (based on user identity information from the MAS) and assume the role of ″vouch for″ server for resources in its domain.

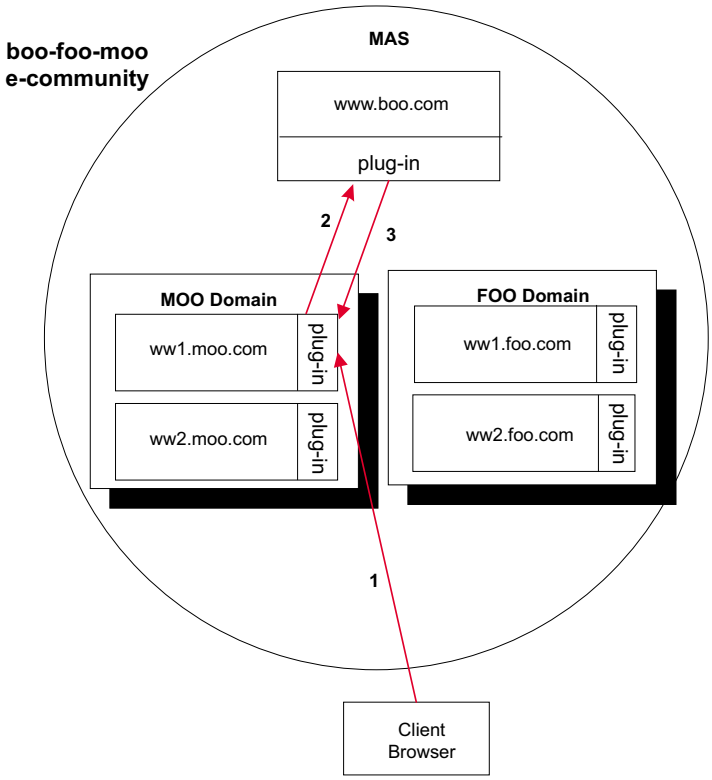

Figure 7. Logging into an e-community.

The example above shows two domains, MOO domain and FOO domain, that exist within an e-community. The following processes take place the first time a user logs on to a secure Web site within the e-community:

1. The user requests access to a resource on the Web server ww1.moo.com. The plug-in intercepts the request and confirms that ww1.moo.com is configured as part of the boo-foo-moo e-community. The MAS server in the e-community is identified from the ww1.moo.com configuration.
2. The request is passed to the MAS - www.boo.com. The MAS authenticates the request on behalf of ww1.moo.com and issues a ″vouch for″ token which becomes the users e-community identity. The user identity information in the token is encrypted.
3. The MAS sends the ″vouch for″ token to ww1.moo.com. ww1.moo.com treats this ″vouch for″ token as proof that the user has successfully authenticated to the MAS and can now access the requested resource based on normal authorization controls.

Understanding the e-community cookie

- The e-community cookie is a domain-specific cookie set by one plug-in, stored in the memory of the user's browser, and transmitted to other plug-in instances (in the same domain) in subsequent requests.
- The domain-specific cookie contains the name of the ″vouch for″ server, the e-community identity, a location (URL) of the ″vouch for″ server and functionality, and a lifetime value. The cookie contains no user information.
- The e-community cookie allows servers in participating domains to request ″vouch for″ information locally. The e-community cookie for the domain where the MAS resides plays a less significant role.
- The cookie has a lifetime (timeout) value that is set in the `pdwebpi.conf` configuration file. This lifetime value specifies how long a remote server is able to provide ″vouch for″ information for the user. When the cookie lifetime has expired, the user must be redirected to the MAS for authentication.
- The cookie is cleared from memory when the browser is closed. If the user logs out of a specific domain, the e-community cookie is overwritten as empty. This action effectively removes it from the browser.

Understanding the ″Vouch For″ request and reply

The e-community ″vouch for″ operation requires dedicated functionality accessed through two specially constructed URLs: the ″vouch for″ request and the ″vouch for″ reply. These URLs are constructed during the e-community ″vouch for″ HTTP re-directs based on the configuration information in `pdwebpi.conf`.

The ″vouch for″ request

The ″vouch for″ request is triggered when a user requests a resource from a target server (configured for e-community) that contains no credential information for that user. The server sends an HTTP re-direct to the ″vouch for″ server (either the MAS or a server identified in an e-community cookie).

The ″vouch for″ request contains the following information:

`https://`*`vouch_for_server`*`/pkmsvouchfor?`*`ecommunity_name`*`&`*`target_url`*

The receiving server checks the *ecommunity_name* to validate the e-community identity. The receiving server uses the *target_url* in the ″vouch for″ reply to re-direct the browser back to the originally requested page.

The **pkmsvouchfor** ″vouch for″ URL is configurable.

For example:

`https://www.boo.com/pkmsvouchfor?companyABC&https://ww2.foo.com/index.html`

The ″vouch for″ reply

The ″vouch for″ reply is the response from the ″vouch for″ server to the target server.

The ″vouch for″ reply contains the following information:

`https://`*`target_url`*`?PD-VFHOST=`*`vouch_for_server`*`&PD-VF=`*`encrypted_token`*

The **PD-VFHOST** parameter identifies the server that performed the ″vouch for″ operation. The receiving (target) server uses this information to select the correct

key required to decrypt the ″vouch for″ token (PD-VF). The **PD-VF** parameter represents the encrypted ″vouch for″ token.

For example:

```
https://ww2.foo.com/index.html?PD-VFHOST=www.boo.com&PD-VF=3qhe9fjkp...ge56wgb
```

Understanding the ″Vouch For″ token

In order to achieve cross domain single signon, some user identity information must be transmitted between servers. This sensitive information is handled using a re-direct that includes the identity information encrypted as part of the URL. This encrypted data is called a ″vouch for″ token.

- The token contains the ″vouch for″ success or failure status, the user's identity (if successful), the fully qualified name of the server that created the token, the e-community identity, and a creation time value.
- The holder of a valid ″vouch for″ token can use this token to establish a session (and set of credentials) at a server without explicitly authenticating to that server.
- The token is encrypted using a shared triple-DES secret key so that its authenticity can be verified.
- Encrypted token information is not stored on the browser.
- The token is only passed once. The receiving server uses this information to build user credentials in its own cache. The server uses these credentials for future requests by that user during the same session.
- The token has a lifetime (timeout) value that is set in the `pdwebpi.conf` configuration file. This value can be very short (seconds) to reduce the risk of a re-play attack.

Encrypting the ″Vouch For″ token

Access Manager Plug-in for Web Servers must encrypt the authentication data placed in the token using a key generated by the **pdwpi-cdsso-key-gen** utility located in the `/bin` directory. You must ″synchronize″ this key by sharing the key file with each plug-in server in each participating domain. Each participating plug-in server in each domain needs to use the same key.

Note: The creation and distribution of key files is not a part of the Access Manager e-community process. You must manually and securely copy keys to each participating server.

The **pdwpi-cdsso-key-gen** utility requires that you specify the location (absolute pathname) of the key file when you run the utility:

UNIX:

```
# pdwpi-cdsso-key-gen absolute_pathname
```

Windows:

```
MSDOS> pdwpi-cdsso-key-gen absolute_pathname
```

The encryption keys are configured in the **[ecsso-domain-keys]** stanza of the `pdwebpi.conf` configuration file. Details of this are covered in the next section, "Configuring an e-community" on page 88

Configuring an e-community

This section reviews all the configuration parameters required for an e-community implementation. These parameters are located in the pdwebpi.conf file. You must carefully configure this file for each participating plug-in in the e-community.

Enabling and Disabling e-Community Members
The **[common-modules]** stanza in the pdwebpi.conf configuration file defines the use of all authentication methods. To enable a plug-in server to operate within an e-community, assign the term 'ecsso' to the **authentication** and **post-authzn** parameters as in the following:

```
[common-modules]
authentication = ecsso
post-authzn = ecsso
```

The **[modules]** stanza in the pdwebpi.conf configuration file defines all available authentication mechanisms and their associated shared library names. Ensure that the entry for e-community SSO exists; that is:

```
[modules]
ecsso = pdwpi-ecsso
```

e-community-name
The **e-community-name** parameter identifies the name of the e-community the server belongs to. For example:

```
[ecsso]
e-community-name = companyABC
```

The **e-community-name** value must be the same for all members of an e-community.

is-master-authn-server
This parameter identifies whether this server is the MAS or not. Values include **yes** or **no**. The parameter would be set as follows for the e-community MAS:

```
[ecsso]
is-master-authn-server = yes
```

Multiple plug-ins can be configured to act as master authentication servers and then placed behind a load balancer. In this scenario, the load balancer is ″recognized″ as the MAS by all other plug-in servers in the e-community.

If **is-master-authn-server** is set to ″yes″ then this server will accept vouch for requests from other plug-in instances whose **e-community-name** is the same and whose domain keys are listed in the **[ecsso-domain-keys]** stanza.

master-authn-server
If the **is-master-authn-server** parameter is set to ″no″, then the **master-authn-server** parameter must be uncommented and specified. The parameter identifies the fully qualified domain name of the e-community MAS. For example:

```
[ecsso]
 master-authn-server = www.boo.com
```

master-http-port
Assign the port number the master authentication server uses to receive HTTP requests. If the port number is not the standard port 80 then the non-standard port number must be specified here.

```
[ecsso]
master-http-port = port_number
```

master-https-port
Assign the port number the master authentication server uses to receive HTTPS requests. If the port number is not the standard port 443 then the non-standard port number must be specified here.

```
[ecsso]
master-https-port = port_number
```

vf-token-lifetime
This parameter sets the lifetime timeout value (in seconds) of the ″vouch for″ token. This value is checked against the creation time stamped on the cookie. The default value is 180 seconds. You must take into account clock skew between participating servers. By default the parameter is set as:

```
[ecsso]
vf-token-lifetime = 180
```

vf-url
This parameter specifies the ″vouch for″ URL The value must begin with a forward-slash (/). The default setting value is: .

```
[ecsso]
vf-url = /pkmsvouchfor
```

You can also express an extended URL:

```
vf-url = /ecommA/pkmsvouchfor
```

ecsso Domain Keys
Defined in the **[ecsso-domain-keys]** stanza of the configuration file are the location of the key files required for encrypting and decrypting tokens between the MAS and participating servers in remote domains. Configuration of the MAS involves defining the keys for each domain for which it is the master. Configuration of e-community members other than the MAS involves defining the key for the domain and for the MAS. You must specify fully qualified domain names for the servers and absolute path names for the key file locations.

The following MAS configuration example provides the MAS with key files for communicating with two remote domains:

```
 [ecsso-domain-keys]
 moo.com = /abc/xyz/moo-boo.key
 bar.com = /abc/xyz/foo-boo.key
```

Configuration for servers in the domains involves specifying the MAS domain and the corresponding key used to exchange information with the MAS. A key is also required for data exchange between servers in the domain. For example the **[ecsso-domain-keys]** stanza for a server in a domain participating in an e-community may look like this:

```
[ecsso-domain-keys]
#the key for data exchange between the MAS (boo.com) and the moo.com domain servers
boo.com = /abc/xyz/moo-boo.key
#the key for data exchange between servers in the moo.com domain
moo.com = /abc/xyz/moo.key
```

Configuring e-community single signon - an example

In the following example there are two e-communities configured – foo-moo and bar-tar – with a single MAS authenticating the requests for both communities.

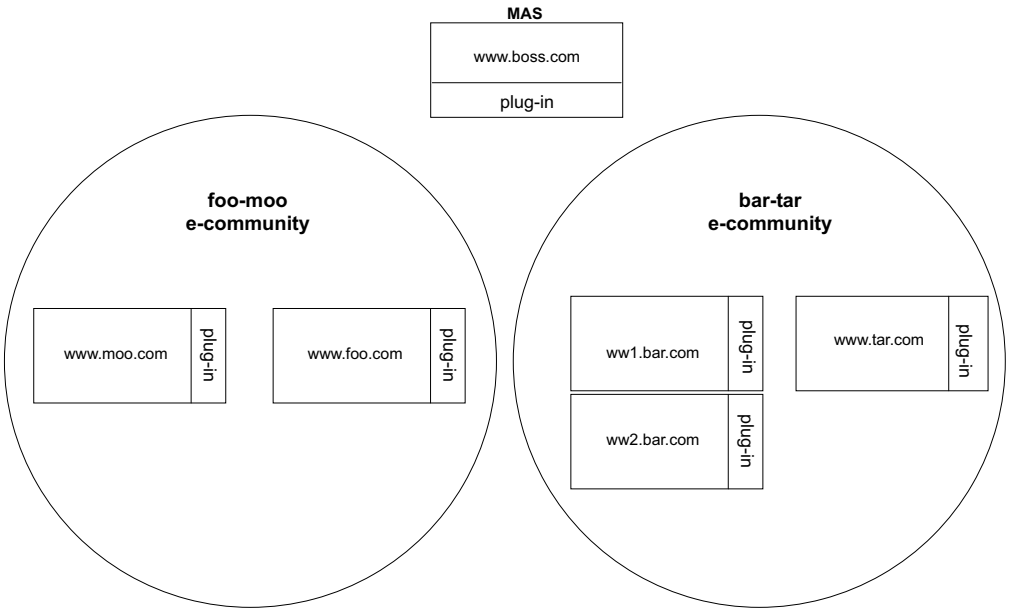

Figure 8. e-Community single signon configuration example

The following conditions apply for this example:

- www.boss.com is the MAS for both e-communities.
- Two distinct domains (one server in each domain for simplicity) exist within the foo-moo e-community – moo.com and foo.com. Users accessing one of these domains can access the other without the need to re-authenticate as all access is granted via the MAS.
- The bar-tar e-community contains two distinct domains – bar.com and tar.com. Users accessing one of these domains can access the other without the need to re-authenticate.
- Users accessing one of the bar.com servers can access the other using a ″vouch for″ token. Single signon in this case is achieved without the need for the MAS to grant access.

In the above example the following configuration conditions apply:

Configuration of the MAS – www.boss.com

As the MAS is the control center for more than one e-community, two distinct instances of the ecsso module need to be configured and the e-community names which the MAS controls need to be defined. The MAS needs to have specified all the keys of the main domains within all the communities it controls. The following configuration conditions apply:

```
[modules]
ecsso1 = pdwpi-ecsso-module
ecsso2 = pdwpi-ecsso-module

[common-modules]
authentication = ecsso1
authentication = ecsso2
```

```
post-authzn = ecsso1
post-authzn = ecsso2

[ecsso1]
e-community-name = foo-moo
is-master-authn-server = yes
.....etc

[ecsso2]
e-community-name = bar-tar
is-master-authn-server = yes
.....etc

[ecsso1-domain-keys]
# one key for each domain the MAS controls
moo.com = /abc/bosskeys/boss-moo.key
foo.com = /abc/bosskeys/boss-foo.key
tar.com = /abc/bosskeys/boss-tar.key
bar.com = /abc/bosskeys/boss-bar.key
```

Configuration of www.moo.com

```
[modules]
ecsso = pdwpi-ecsso-module

[common-modules]
authentication = ecsso

post-authzn = ecsso

[ecsso]
e-community-name = foo-moo
is-master-authn-server = no
master-authn-server = www.boss.com
.....etc

[ecsso-domain-keys]
#key for encrypting/decrypting data between servers in the moo.com domain
moo.com = /abc/moo-keys/moo.key
#key for encrypting/decrypting data between servers in the moo.com domain and the MAS
boss.com = /abc/moo-keys/boss-moo.key
```

Configuration of www.foo.com

The configuration parameters for achieving single signon to www.foo.com will be identical to those configured for www.moo.com except the domain keys will be different. Domain keys configuration for www.foo.com would be as follows:

```
[ecsso-domain-keys]
#key for encrypting/decrypting data between servers in the foo.com domain
foo.com = /abc/foo-keys/foo.key
#key for encrypting/decrypting data between servers in the foo.com domain and the MAS
boss.com = /abc/foo-keys/boss-foo.key
```

Configuration of www.tar.com

```
[modules]
ecsso = pdwpi-ecsso-module

[common-modules]
authentication = ecsso

post-authzn = ecsso

[ecsso]
e-community-name = tar-bar
is-master-authn-server = no
master-authn-server = www.boss.com
.....etc
```

```
[ecsso-domain-keys]
#key for encrypting/decrypting data between servers in the tar.com domain
tar.com = /abc/tar-keys/tar.key
#key for encrypting/decrypting data between servers in the tar.com domain and the MAS
boss.com = /abc/tar-keys/boss-tar.key
```

Configuration of ww1.bar.com

The e-community single signon configuration for ww1.bar.com is identical to that of www.tar.com. Two keys are required, one for encrypting/decrypting data between the MAS and the bar.com domain and a key for encrypting/decrypting data between servers within the bar.com domain (i.e. ww1.bar.com and ww2.bar.com in this example).

```
[ecsso-domain-keys]
bar.com = /abc/bar-keys/bar.key
boss.com = /abc/bar-keys/boss-bar.key
```

Configuration of ww2.bar.com

The definition of keys for ww2.bar.com will be identical to that for ww1.bar.com.

```
[ecsso-domain-keys]
bar.com = /abc/bar-keys/bar.key
boss.com = /abc/bar-keys/boss-bar.key
```

Appendix A. pdwebpi.conf reference

Table 29. General configuration parameters

General	
Parameter	**Description**
[pdweb-plugins]	
virtual-host	Identification of subordinate stanzas that contain configuration information about specific virtual hosts.
unprotected-virtual-host	Identifies virtual hosts that the plug-in does not provide security for. The plug-in permits access to these virtual hosts without performing authentication and authorization on the request. An entry in the log file is created when unprotected virtual hosts are accessed.
web-server	Identifies the type of web server in use. The acceptable values are: • iis for Microsoft Internet Information Services • ihs for IBM HTTP Server • iplanet for iPlanet Web Server This parameter is set automatically during installation.
windows-file-system	Indicates to the Authorization Server that precautions should be taken to avoid security issues related to URIs representing Windows file system resources. Set to **true**, any access to a URI with path elements like Windows 2000 short path names are forbidden. In particular path elements ending with *~digit* are rejected. On Windows systems this parameter is set to **true** by default. On UNIX systems it is set to **false**. This parameter may be overridden on a per-virtual host basis by specifying it in the appropriate **[***virtual_host***]** stanza.
case-sensative	Tells the Authorization Server how to handle the case of URIs. Set to **false**, URIs are converted to lower case when constructing the corresponding Access Manager object name against which an authorization decision is made. On UNIX systems this parameter is set to **true**. On Windows systems it is set to **false**. When the **windows-file-system** parameter is set to **true** and **case-sensitive** is not defined, URIs are converted to lower case by default. Note that /PDWebPI/*branch* portion of the object name is not so translated. This parameter may be overridden on a per-virtual host basis by specifying it in the appropriate **[***virtual_host***]** stanza.

© Copyright IBM Corp. 2002

Table 29. General configuration parameters (continued)

General	
Parameter	**Description**
utf8-url-support-enabled	Controls the code page under which URLs are interpreted when building a corresponding Access Manager Protected Object Name. Set to **true**, URIs presented to the Authorization Server are assumed to be encoded in UTF8 and are converted to the code page under which the Authorization Server is running before being used to construct an Access Manager Protected Object Name. Set to **false**, URIs presented to the Authorization Server are assumed to be encoded in the code page under which the Authorization Server is running. Set to **auto**, each URI is examined for multi-byte UTF8 sequences. If found the URI is assumed to be encoded in UTF8. If invalid UTF8 character sequences are detected, the URI is assumed to be in the code page under which the Authorization Server is running. This parameter can be overridden on a per-virtual host basis by specifying it in the appropriate **[***virtual_host***]** stanza.
log-file	Identifies the filename and path of the log file where all Authorization Server tasks are captured.
logs	Specifies the number of log files to create before re-using the first log file.
log-entries	Specifies the number of log entries to be written before rolling over to a new log file.
mpa-enabled	Multiplexing Proxy Agents (MPA) are gateways that accommodate multiple client access. A single authenticated channel to the origin server is established and all client request and response communications are sent via this channel. Set to **true**, MPA capability is enabled. Set to **false**, MPA capability is disabled. This parameters can be overridden on a per-virtual host basis by defining it in the **[***virtual_host***]** stanza.
mpa-protected-object	Defines the MPA object against which the authorization decision is made. This parameters can be overridden on a per-virtual host basis by defining it in the **[***virtual_host***]** stanza.
user	On UNIX systems this parameter defines the user name for Policy Manager and Authorization Server processes.
group	On UNIX systems this parameter defines the group name for Policy Manager and Authorization Server processes.
[module-mgr]	

Table 29. General configuration parameters (continued)

General	
Parameter	**Description**
path	The path containing the module shared library files. More than one path entry is permitted as the plug-in will search all entries.
[wpiconfig]	
server-type	Set during configuration to aid with unconfiguration.
install-dir	Set during configuration to aid with unconfiguration.
vhosts	Set during configuration to aid with unconfiguration.

Table 30. Authentication configuration parameters

Authentication	
Parameter	**Description**
[modules]	
module_name = *shared_library_name*	Declares the available authentication methods and the associated library.
acctmgmt	Account Management
BA	Basic Authentication
cert	Certificate
failover	Failover
forms	Forms
ip-addr	IP Address
iv-headers	IV Headers
session-cookie	Session Cookie
ssl-id	SSL Id
tag-value	tag value
http-hdr	HTTP Header
token	Token
[common-modules]	
authentication	Specifies the methods to use for user authentication.
session	Specifies the methods to use for maintaining session state.
post-authzn	Specifies the methods to use for post-authorization processing.
[authentication-levels]	

Table 30. Authentication configuration parameters (continued)

Authentication	
Parameter	**Description**
level = *module_name*	The **[authentication-levels]** stanza defines step-up authentication levels as well as the ordering of authentication methods defined in the **[modules]** stanza. An authentication method defaults to a level of one when no entry for it is defined. Authentication order is determined as the highest authentication level down to the lowest authentication level for the authentication methods defined. If an authentication level is shared by several authentication methods the sub-order is determined by the order in which the modules appear within the **[modules]** stanza.
[authentication-mechanisms]	
passwd-cdas **passwd-ldap** **passwd-uraf** **token-cdas** **cert-ssl** **cert-cdas** **http-request** **cdsso** **passwd-strength** **cred-ext-attrs**	List of supported, additional authentication mechanisms and associated shared libraries which plug-in to the authentication subsystem of Access Manager. .
[BA]	
basic-auth-realm	Declares the realm name that will appear on the dialogue presented to the user during basic authentication log on.
[failover]	
failover-cookies-keyfile	Declares the path to the key file used to encrypt and decrypt credential data in the failover cookie.
failover-cookies-lifetime	The valid lifetime of a failover cookie in minutes.
enable-failover-cookie-for-domain	Enable/disable the failover cookie for the extent of the whole domain.
[ltpa]	
ltpa-keyfile	Full path name of the LTPA key file.
ltpa-stash-file	Location of the password stash file
ltpa-password	The password to use in lieu of stash file.
ltpa-lifetime	The lifetime in seconds of the LTPA cookie.
[forms]	
login-form	The file name of the logon form.
[tag-value]	
cache-definitions	Enables or disables caching of tag value definitions which are attached to the object space. When caching is enabled, the plug-in requires restarting each time changes are made to tag/value definitions.
cache-refresh-interval	The refresh interval in seconds for the cache of definitions.

Table 30. Authentication configuration parameters (continued)

Authentication	
Parameter	**Description**
[token-card]	
token-login-form	The file name for the token logon page.
next-token-form	Defines the form displayed to the user client to request the next token. The client is requested to enter another token when the server can not successfully authenticate the user from the first.
[http-hdr]	
header	The header name passed to the Cross Domain Authentication Service (CDAS) for authentication.
[iv-headers]	
accept	List of headers to accept as proof of authentication from a proxy. The valid options are: • **all** - accepts all header types. • **iv-creds** - user credential information. • **iv-user** - Short user name. • **iv-user-l** - long user name. • **iv-remote-address** - IP address of the client.
generate	List of headers to generate when forwarding a request from a proxy. The valid options are: • **all** - generates all header types. • **iv-creds** - user credential information. • **iv-user** - Short user name. • **iv-user-l** - long user name. • **iv-remote-address** - IP address of the client.
[acctmgmt]	
password-change-form	The form displayed when a user requests a change of password.
password-change-form-uri	The URI accessed when a user requests a change of password.
password-change-uri	The URI destination after password change.
password-change-success	The page displayed when a user completes a change of password successfully.
password-change-failure	The page displayed when a user fails to logon successfully.
logout-uri	The URI destination after user logout.
logout-success	The page displayed when a user successfully logs out.
help-uri	The location of the help page.
help-page	The file name of the help page displayed when a user requests help.
[ecsso]	
e-community-name	The e-community name that appears in ″vouch for″ tokens and requests.

Table 30. Authentication configuration parameters (continued)

Authentication	
Parameter	**Description**
is-master-authn-server	Specifies whether the server is a master or not in the e-community. Set to **yes**, this server accept vouch for requests from other plug-in instances whose domain keys are listed in the **[ecsso-domain-keys]** stanza.
master-authn-server	The name of the master server in an e-community. This parameter is mandatory if **is-master-authn-server** is set to no.
master-http-port	Port number to listen for HTTP requests from the master server.
master-https-port	Port number to listen for HTTPS requests from the master server.
vf-token-lifetime	The vouch for token lifetime in seconds.
vf-url	The vouch for URL.
[inter-domain-keys]	
domain_name = *key_file*	Locations of the key files of other domains participating in the e-community.

Table 31. Sessions configuration parameters

Sessions	
Parameter	**Description**
[sessions]	
max-entries	The maximum number of sessions stored within a single instance of a session module. Maximum number of sessions per session module per virtual host.
timeout	The maximum lifetime of a session in seconds.
inactive-timeout	The length of idle time in seconds required for a session before it times out.
resend-pdwebpi-cookies	Enables or disables the sending of Web Plug-in cookies with each request.
reauth-lifetime-reset	Controls the session lifetime timer. Set to 'yes', the session lifetime timer (i.e. the value set in the timeout parameter) is reset upon successful reauthentication. Set to 'no' a reset is not performed for a successful reauthentication.
reauth-grace-period	Sets the amount of time in seconds the client has as a grace period within which to successfully perform reauthentication when the credential would otherwise have expired.

Table 32. LDAP configuration parameters

LDAP	
Parameter	**Description**
[ldap]	
bind-pwd	The password for the Web Plug-in Daemon (set during configuration).

Table 32. LDAP configuration parameters (continued)

LDAP	
Parameter	**Description**
enabled	Enables or disables LDAP communication (set during configuration).
host	The name of the LDAP server (set during configuration).
port	The port number of the LDAP (set during configuration).

Table 33. Proxy configuration parameters

Proxy	
Parameter	**Description**
[ipc]	
number-of-workers	The number of worker threads that handle plug-in requests.
worker-size	The amount of memory pre-allocated for each worker worker thread handling plug-in requests.
cleanup-interval	Time in seconds between each clean-up of memory.
max-session-lifetime	Defines the maximum lifetime of a session.
[proxy]	
error-page	The path to the page displayed on the users browser when an unexpected server error occurs.
acct-locked-page	The path to the page displayed when a user attempts to access a locked account.
retry-limit-reached-page	The path to the page displayed when the allowed maximum number of failed logon attempts has been reached. The maximum number of allowed log on failures is set in LDAP using the **policy** command..

Table 34. Authorization API configuration parameters

Authorization API	
Parameter	**Description**
[aznapi-configuration]	
Audit and Logging Parameters and Configuration	
logsize	The size in bytes over which a new log file is created. If set to 0, a new log file is not created. If set to a negative number, a new log file is created daily regardless of size.
logflush	The interval in seconds at which the logs are flushed. The maximum value is 21600 (6 hours).
logaudit	Enable / disable audit logging.
auditlog	The name of the audit file.

Table 34. Authorization API configuration parameters (continued)

Authorization API	
Parameter	**Description**
auditcfg	Enable/disable component specific audit records. That is; auditcfg = authn — Capture authentication events. auditcfg = azn — Capture authorization events.
db-file	Location of the ACL database cache file.
cache-refresh-interval	The interval in seconds, between checks for updates to the master authorization server.
listen-flags	Enable/disable flags for the reception of policy cache update notifications.
Authorization API Service Definitions	
[aznapi-entitlement-services]	
service_id	Each stanza entry defines different types of aznAPI service. For more information refer to the Authorization API programmers guide.
AZN_ENT_EXT_ATTR	This is a system level parameter that should not be changed. It allows the use of extended attributes on the object space.
[aznapi-admin-services]	
name **=** *shared_library_name* **-pobj** *object_space* **&** *args*	Configuration for the admin service. Supported arguments are: -r the protected objectspace root -d the Web server's document root -q the program to use for query_contents -v the virtual host ID (this parameter is optional

Table 35. Web server specific configuration parameters

Web Server Specific	
Parameter	**Description**
[ihs]	
query-contents	Specifies the query contents program to use for browsing the IBM HTTP Server Web space by the 'pdadmin> object list' command. This parameter can be overridden on a per branch basis by specifying a value for it in a stanza named **[ihs:***branch***]** e.g. [ihs:/PDWebPI/foo.bar.com]
doc-root	Specifies the documentation root which provides the Web space browse capability needed for performing 'pdadmin> object list' commands. This parameter is set by the configuration utility when setting up virtual hosts - it is specified on a per-policy branch basis in an **[ihs:***branch***]** stanza e.g. [ihs:/PDWebPI/foo.bar.com]
[iis]	

Table 35. Web server specific configuration parameters (continued)

Web Server Specific	
Parameter	**Description**
query-contents	Specifies the query contents program for browsing the IIS Web space by pdadmin. This parameter can be overridden on a per branch basis by specifying a value for it in a stanza named **[iis:***branch***]** e.g. [iis:/PDWebPI/foo.bar.com]
post-data-required	Defines a list of forms for which the submitted POST data is required for processing by the Authorization Server. For example a logon form. These parameters may NOT be overridden on a per-virtual host basis.
log-file	Defines the log file for error and trace messages from the IIS plug-in which are kept separate from the Authorization Server's log file in order to ensure consistency of the files. If specified as a relative path the location is relative to the log sub-directory of the install directory. If specified as an absolute path the absolute path is used.
[iis:minimum-post-data]	
form_uri = *minimum_bytes_of_post_data_required*	Defines the amount of post data for a particular form in cases when large amounts of post data are required. For example; `/token.form = 20000` Indicates that at least 20000 bytes of POST data are required by the Authorization Server when processing submissions to /token.form. These values cannot be specified on a per-virtual host basis.
[iplanet]	
query-contents	Specifies the query contents program for browsing the iPlanet Web space by pdadmin. This parameter can be overridden on a per branch basis by specifying a value for it in a stanza named **[iplanet:***branch***]** e.g. [iplanet:/PDWebPI/foo.bar.com]
doc-root	Specifies the documentation root which provides the Web space browse capability needed for performing 'pdadmin> object list' commands. This parameter is set by the configuration utility when setting up virtual hosts - it is specified on a per-policy branch basis in an **[iplanet:***branch***]** stanza e.g. [iplanet:/PDWebPI/foo.bar.com]

Appendix B. Authentication methods quick reference

Table 36. Plug-in authentication method/module reference

Authentication method/module	Description
BA `pdwpi-ba-module`	Basic Authentication authentication module. May also be configured as a session and post-authorization module.
forms `pdwpi-forms-module`	HTML Forms authentication module. Authenticates using a username and password submitted via a form. When in use, this module must also be configured as a post-authorization module.
ip-addr `pdwpi-ipaddr-module`	Client IP Address authentication module. Provides authentication based solely on the client's IP address. A http-request authentication mechanism must be provided by the customer to map the IP address information to an Access Manager principal. May also be configured as a session module.
http-hdr `pdwpi-httphdr-module`	HTTP Header authentication module. Provides authentication based solely on the value of a nominated HTTP header in the request. An http-request authentication mechanism must be provided by the customer to map the header information to an Access Manager principal. May also be configured as a session module.
token `pdwpi-token-module`	Token authentication module. Access Manager Plug-in for Web Servers supports authentication via a token passcode supplied by the client. This authentication uses a two factor log on based on RSA SecureID fobs. When in use, must also be configured as a post-authorization module.
cert `pdwpi-certificate-module`	Client certificate authentication module. The subject DN of the client certificate is mapped by the cert-ssl authentication mechanism to an Access Manager principal name. The cert-ssl authentication mechanism requires that the subject DN of the client certificate map directly to the DN of an Access Manager user in the user registry. This module ignores requests to authenticate requests that did not arrive over an SSL session and so can be safely configured for virtual hosts that handle authorization of both HTTP and HTTPS requests.

© Copyright IBM Corp. 2002

Table 36. Plug-in authentication method/module reference (continued)

Authentication method/module	Description
failover `pdwpi-failover-cookie-module`	Failover Cookie authentication module. This module accepts a failover cookie to authenticate a user. When in use, this module must also be configured as a post-authorization module.
iv-headers `pdwpi-iv-headers-module`	IV Headers authentication module. Provides authentication based on the values of the iv-user, iv-user-l, iv-creds, or iv-remote-address HTTP header in the request. This is useful for single-signing on to Access Manager Plug-in for Web Servers when a user has already authenticated to a front-end proxy server. In order to be trusted, the request must have arrived via an authenticated session with a front-end proxy server (e.g. a WebSEAL junction). The proxy must authenticate as a user with Proxy ([PDWebPI]p) permission on the protected object space branch of the virtual host being accessed. For authentication using iv-remote-address header, a http-request authentication mechanism must be provided by the customer to map the IP address information to an Access Manager principal. This module may also be configured as a post-authorization module.
ecsso `pdwpi-ecsso-module`	e-Community Single signon authentication module. This module must be configured as an authentication module for virtual hosts other than the master authentication server that are participating in the e-community. When in use, this module must also be configured as a post-authorization module.
unauth `pdpwi-unauth-module`	Unauthenticated user authentication module. This module is listed here for completeness. It is implicitly always configured as the lowest precedence authentication module and is used to generate a credential for unauthenticated users.

Table 37. Plug-in session module reference

Module	Description
BA `pdwpi-ba-module`	Basic Authentication session module. Use the Basic Authentication Authorization header value as a session key. When in use, must also be configured as an authentication module. May also be configured as a post-authorization module.

Table 37. Plug-in session module reference (continued)

Module	Description
ip-addr `pdwpi-ipaddr-module`	IP Address session module. Uses an authenticated client IP address as the session key. When in use, it must also be configured as an authentication module.
http-hdr `pdwpi-httphdr-module`	HTTP Header session module. Uses an authenticated HTTP header as the session key. When in use, it must also be configured as an authentication module.
session-cookie `pdwpi-sesscookie-module`	Session Cookie session module. This module generates and accepts cookies for use in identifying sessions. Generally used only as a low priority session identification mechanism.
ssl-id `pdwpi-sslsessid-module`	SSL Session ID session module. Uses the SSL Session ID as a session key. Note that although this module is provided in the Windows distribution of Access Manager Plug-in for Web Servers, the Microsoft Internet Information Services Web Server does not provide SSL Session ID information to the plug-in so SSL Session IDs cannot be used as session keys for IIS.

Table 38. Plug-in post-authorization module reference

Module	Description
forms `pdwpi-forms-module`	HTML Forms post-authorization module. This module handles the submission of the form data during an HTML Forms based log on. When in use, it must also be configured as an authentication module.
BA `pdwpi-ba-module`	Basic Authentication post-authorization module. When configured as a post-authorization module, the BA module removes any unauthenticated Basic Authentication Authorization headers from the request. The Basic Authentication module can also be configured as an authentication and session module.
token `pdwpi-token-module`	Token post-authorization module. Access Manager Plug-in for Web Servers supports authentication via a token passcode supplied by the client. This authentication uses a two factor log on based on RSA SecureID fobs. When in use, the token module must also be configured as an authentication module.

Table 38. Plug-in post-authorization module reference (continued)

Module	Description
failover `pdwpi-failovercookie-module`	Failover Cookie post-authorization module. This module generates a failover cookie for the client. When in use, the failover cookie module must also be configured as an authentication module.
iv-headers `pdwpi-iv-headers-module`	IV Headers post-authorization module. This module inserts user identity information as IV headers in to the request before allowing the request to be handled by the Web server. This is useful for providing single sign on to an application hosted by the Web server. The headers that can be added are iv-user, iv-user-l, iv-groups, iv-creds, iv-remote-address. This module may also be configured as an authentication module.
tag-value `pdwpi-tag-value-module`	Tag/Value post-authorization module. This module inserts additional extended attributes from the users credential as HTTP headers in the request before allowing the request to be handled by the Web server. These extended attributes typically correspond to user attributes from the user registry.
acctmgmt `pdwpi-acct-mgmt-module`	Account Management post-authorization module. This module provides the logout (/pkmslogout), change password (/pkmspasswd), help (/pkmshelp) features.
ltpa `pdwpi-ltpa-module`	LTPA Cookie post-authorization module. This module inserts a WebSphere Application Server (WAS) Lightweight Third Party Authentication (LTPA) cookie into the request before allowing the request to be handled by the Web server. This provides single sign on to a WAS being hosted by the Web server.
ecsso `pdpwi-ecsso-module`	e-Community Single Sign–On post-authorization module. All virtual hosts participating in an e-community must have the ecsso module configured as a post-authorization module. This module must also be configured as an authentication module for all participants other than the master authentication server.

Appendix C. Command quick reference

Table 39. Plug-in command reference

Command	Description

pdwebpi_start	Starts and stops the plug-in process on UNIX installations.
	
	The valid options are:
	`pdwebpi_start {start|stop|restart|status}`
	
	To stop the plug-in and then restart it, use:
	`# pdwebpi_start restart`
	
	The **pdwebpi_start** command is located in the following directory:
	`/opt/pdwebpi/sbin/`
	
	To start and stop plug-in Windows installations identify the plug-in process in the Services Control Panel and use the appropriate control buttons.

pdwpi-cdsso-key-gen	Creates a key file used for encrypting and decrypting plug-in data such as failover cookie information and ″Vouch For″ tokens. **Usage:** `./pdwpi-cdsso-key-gen` *`key_file_name_to_create`* The **pdwpi-cdsso-key-gen** command is located in the `/bin` directory.
pdwpi-version	Lists the version and copyright information for the installation. The **pdwpi-version** command is located in the `/bin` directory.
pdwpicfg	Starts the utility for configuring and unconfiguring the plug-in. The **pdwpicfg** command is located in the `/bin` directory.

© Copyright IBM Corp. 2002

Appendix D. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corp. 2002

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

AIX
DB2
IBM
IBM(logo)
Java
OS/390

SecureWay
Tivoli
Tivoli (logo)
Universal Database
WebSphere
z/OS
zSeries

Microsoft, Windows, are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Index

A

ACL policies 63
ACL policy, default 64
AIX
  installation on 8
  removal from 13
API service 27
applying unauthenticated HTTPS 76
architecture 1
auditing 24
authentication
  background 29
  configuration overview 43
  forms 47
  goals of 4
  methods 33
    order of 33
    quick reference 103
  network based POP policy 73
  parameters 95
  step-up 69
  understanding 4
  with Basic Authentication 46
  with certificates 48
  with failover cookies 51
  with HTTP headers 55
  with IP addresses 56
  with IV headers 53
  with tokens 50
authentication mechanism
  Basic Authentications 46
  certificates 49
  forms 47
  HTTP header 56
  tokens 50
  with IP addresses 57
  with IV headers 54
authentication mechanisms 43
authentication methods 44
authentication modules
  quick reference 103
authentication strength POP
  IP address for sessions 68
authorization server
  configuration 18

B

Basic Authentication 40, 46
books
  feedback xii
  online xii
  ordering xii

C

cache database 24
CDAS authentication parameters 43
certificates 48
cleanup-interval parameter 18
commands
  change of password 45
  help 45
  logout 45
components 1
configuration
  API service 27
  audit logs 24
  authentication 32
  authentication for virtual hosts 36
  authentication methods 44
  authentication overview 43
  Authorization Server 18
  Basic authentication for authentication 46
  Basic Authentication for sessions 40
  cache database 24
  certificate authentication 48
  default 44
  e-community single signon 88
  failover cookies for authentication 51
  forms for authentication 47
  HTTP headers for authentication 55
  HTTP headers for sessions 42
  IP address for authentication 56
  IP address for sessions 42
  IV headers for authentication 53
  logs 24
  of pdwebpimgr.conf 16
  parameters
    authentication 95
    authorization API 99
    general 93
    LDAP 98
    proxy 99
    sessions 98
    web server specific 100
  pdwebpi.conf 15
  plug-in 15
  post-authorization 36
  server specific 22
  session cookies for sessions 41
  session/credentials cache 38
  SSL session ID for sessions 40
  stanzas 93
  tag value for post-authorization 57
  token response pages 51
  tokens for authentication 50
  virtual hosts 19
credential
  acquisition 5
Customer Support xvii

D

disk and memory requirements 7

E

e-community single signon
  configuring 88

© Copyright IBM Corp. 2002

e-community single signon *(continued)*
cookie 86
encrypting vouch for token 87
features and requirements 84
introducing 83
process flow 84
e-mail contact xvii
EPAC 5
error messages 17
Extended Privilege Attribute Certificate (EPAC) 5

F

failover cookies 51
features 3
feedback about publications xvii
forms authentication 47

H

HTML response forms 48
HTTP error messages 17
HTTP headers 42, 55

I

id parameter 18
IHS
installation on 8
removal from 13
required version 7
IIS
installation on 11
removing from 12
required version 7
installation
on AIX/IHS 8
on Solaris Operating Environment/iPlanet 10
on Windows/IIS 11
software prerequisites 7
installation directory 16
IP addresses 42, 56
ipc stanza 18
iPlanet
installation on 10
required version 7
iPLanet
removal from 13
IV headers 53

L

LDAP, configuration parameters 98
local authentication parameters 43
log on
forcing 75
log on, prompting for 44
logging 24
logon policy 65

M

macro support 17
manuals
feedback xii
manuals *(continued)*
online xii
ordering xii
max-entries parameter 38
max-session-lifetime 18
memory requirements 7
MPAs 59
Multiplexing Proxy Agents 59

N

network based authentication POP policy 73
number-of-workers parameter 18

O

online publications xvi
ordering publications xvi

P

password policy 66
pdweb-plugin stanza 19
pdwebpi.conf 15
pdwebpimgr.conf 16
permissions, ACL 63
permissions, WebDAV 64
pkmshelp 45
pkmslogout 45
pkmspasswd 45
plug-in
authentication 4, 29
configuration 18
disk and memory requirements 7
features 3
HTTP error messages 17
installation 7
installation directory 16
macro support 17
security policy 3
software prerequisites 7
starting and stopping 16
supported platforms 7
plug-in process flow 1
policy
ACL 64
authentication strength POP 68
controlling unauthenticated users with 76
logon 65
network based authentication POP 73
password 66
quality of protection POP 74
step-up 68
POP policy
authentication strength - step up 68
network based authentication 73
quality of protection 74
reauthentication 71
post-authorization
with tag value 57
post-authorization processing 36
publications
feedback xii
online xii
ordering xii

Q

Quality of protection POP policy 74

R

realm name, setting 46
reauth-grace-period 39
reauth-lifetime-reset 39
reauthentication 71
Removal
  from AIX/IHS 13
  from Solaris/iPlanet 13
Removing
  from Windows/IIS 12
required software 7
root directory 16

S

security policy 3
session cache 38
session cookies 41
session reauthentication reset 39
session state
  managing 37
  with Basic Authentication 40
  with HTTP headers 42
  with IP addresses 42
  with session cookies 41
  with SSL session ID 40
session timeout 39
single signon
  concepts 77
  e-community 83
  to WebSEAL 80
  using failover cookies 81
  using HTTP headers 78
  using LTPA cookies 79
software prerequisites 7
Solaris Operating Environment
  installation on 10
  removal from 13
SSL session ID 40
stanzas, configuration file 93
starting the plug-in 16
step-up 68
stopping the plug-in 16
supported platforms 7
system requirements 7

T

tag value 57
Tivoli Customer Support xvii
token authentication 50
token response pages 51
tokens 50
trace 26

U

unauthenticated users 75
uninstall 12
unprotected-virtual-host parameter 19

V

versions
  of required software 7
virtual hosts
  authentication configuration 36
  authentication parameters 58
  configuring 19
  support for 2
virtual-host parameter 19
vouch for request and reply 86
vouch for token 87

W

Web server versions, required 7
WebDAV permissions 64
WebSEAL 80
Windows
  installation on 11
  removing from 12
worker threads, configuring 18
worker-size parameter 18

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

GC23-4686-00